SIGN LANGUAGE RECOGNITION SYSTEM

by

BASITH K P

(Registration Number: 21352011)

Project report submitted in partial fulfillment of the requirements for the award of the degree of

MASTER OF COMPUTER APPLICATIONS

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF ENGINEERING & TECHNOLOGY

PONDICHERRY UNIVERSITY

May 2023

PONDICHERRY UNIVERSITY

(*A Central University*)

SCHOOL OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

Master of Computer Applications

NAME	:	BASITH K P
REG. NO.	:	21352011
SEMESTER	:	IV
SUBJECT	:	MAIN PROJECT REPORT
GUIDED BY	:	**DR. S. SIVA SATHYA**

BONAFIDE CERTIFICATE

This is to certify that this project work entitled **"Sign Language Recognition System"** is a bonafide record of work done by **Mr. BASUTH K P** (Reg. Number 21352011) in the partial fulfillment for the degree of Master of Computer Applications of Pondicherry University.

This work has not been submitted elsewhere for the award of any other degree to the best of our knowledge.

INTERNAL GUIDE

Dr. S. Siva Sathya
Professor/HOD
Department of Computer Science
School of Engineering & Technology
Pondicherry University
Puducherry – 605 014

HEAD OF THE DEPARTMENT

Dr. S. Siva Sathya
Professor/HOD
Department of Computer Science
School of Engineering & Technology
Pondicherry University
Puducherry – 605 014

Submitted for the Viva-Voce Examination held on:

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

I am greatly thankful to all those who helped me in making this project a successful one.

I express my heartfelt gratitude to my Project Guide **Dr. S.Siva Sathya, Professor/HOD, Department of Computer Science, Pondicherry University, Puducherry**, for her whole - hearted assistance and direction not only for the duration of the project but for the entire duration of the course. I will always remain grateful to her and whose constant care about me has provided a new direction to work.

I express my gratitude to **Dr. T. Chithralekha, Dean, School of Engineering & Technology, Pondicherry University, Puducherry** for providing us the facilities to work with the project.

Finally, I would like to express my regards for all the faculty members of Department of Computer Science and others involved in this project, directly or indirectly. I am also grateful to my parents for their continuous support and encouragement.

Last but not least, the blessings of **God** enabled me to complete the project successfully.

SYNOPSIS

Sign language, as a different form of the communication language, is important to large groups of people in society. While there are many different types of gestures, the most structured sets belong to the sign languages. In sign language, each gesture already has assigned meaning, and strong rules of context and grammar may be applied to make recognition tractable. But only less people know sign language and that makes the speech impaired persons to efficiently interact with society. Here this system's importance comes in picture.

The system detects sign language with python. Using Convolutional Neural Network**,** we have an input image, that is then passed through the network to get an output predicted label. Dense Convolutional Network (DenseNet), which connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections - one between each layer and its subsequent layer - our network has L(L+1)/2 direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers.

The above-mentioned model is using in this proposed work. And this model will use for the mobile app creation which translates sign language in real time.

The main objectives are:

- Build a Sign Language model using an Action Detection powered by Denset121
- Predict sign language in real time.
- Use this model in a Mobile app for sign language translation

TABLE OF CONTENTS

TITLE	PAGE NO.
ACKNOWLEDGEMENT	ii
SYNOPSIS	iii
LIST OF FIGURES	vii
LIST OF ABBREVIATIONS	viii
1. INTRODUCTION	**1-2**
1.1 ABOUT THE PROJECT	1
1.2 PLAN OF REPORT	1
2. PROBLEM DEFINITION & FEASIBILITY ANALYSIS	**3-9**
2.1 INTRODCUTION	3
2.2 PROBLEM DEFINITION	3
2.3 EXISTING SYSTEM	3
2.4 PROPOSED SYSTEM	5
2.4.1. Features	6
2.5 FEASIBILITY ANALYSIS	7
2.5.1 Operational Feasibility	7
2.5.2 Technical Feasibility	8
2.5.3 Economic Feasibility	9
3. SOFTWARE REQUIREMENT SPECIFICATION	**10-21**
3.1 INTRODUCTION	10
3.1.1 Purpose	10
3.1.2 Scope	10
3.2 SYSTEM REQUIREMENTS	11
3.2.1 Hardware Specification	11
3.2.2 Software Specification	12
3.3 USER REQUIREMENTS	12
3.3.1 Functional Requirement	12
3.3.2 Non-Functional Requirement	13
3.3.3 Performance Requirement	13
3.4 EXPLANATION OF TECHNOLOGIES USED	14

3.4.1 Flutter and Dart 14
3.4.2 Convolutional neural network (CNN) 14
3.4.3 Mediapipe Holistic 14
3.4.4 Tensorflow and Tensorflow lite 15
3.4.5 Data Augmentation 15
3.4.6 Deeplearning optimization techniques 15
3.4.7 Natural Language processing 15
3.4.8 Flask API 15
3.5 ALGORITHM DEFINITION 16
3.5.1 Convolutional neural network 16
3.5.2 DensNet121 20
3.5.3 Flask API 20
4. SYSTEM DESIGN 22-27
4.1 INTRODUCTION 22
4.2 USE CASE DIAGRAM 22
4.3 DATA FLOW DIAGRAM 23
4.3.1 Work flow of deep learning model 25
4.4 CLASS DIAGRAM 25
4.5 SEQUENCE DIAGRAM 26
5. CODING, TESTING, AND IMPLEMENTATION 28-36
5.1 INTRODUCTION 28
5.2 EXPERIMENTS AND RESULTS 28
5.2.1 Dataset collection and preprocessing 28
5.2.2 Training and Testing 29
5.2.3 API and Mobile App 32
5.3 IMPLIMENTATION 32
5.4 TESTING 33
5.4.1 Unit Testing 34
5.4.2 Validation Testing 34
5.4.3 Functional Testing 34
5.4.4 GUI Testing 34
5.5 CODING 35
6. CONCLUSION AND FUTURE WORK 37
6.1 Conclusion 37

6.2 Future Work 37

REFERENCES 39

APPENDICES

APPENDIX A: SAMPLE SCREEN SHOTS 40-41

LIST OF FIGURES

FIGURE NO.	FIGURE NAME	PAGE NO.
3.1	CNN Architecture	16
3.2	DenseNet Architectures	17
4.1	Use case diagram	18
4.2	Data flow diagram	27
4.3	Work flow of deep learning model	27
4.4	Class diagram	28
4.5	Sequence Diagram	28
5.1	Input image of sign A	29
5.2	Extracted Features	29
5.3	Model Accuracy curve	30
5.4	Model lose curve	31
5.5	Test Accuracy Curve	31
5.6	Confusion matrix	32
6.1	Prediction On openCV	
6.2	Home page of the APP	
6.3	Tutorial page of APP	
6.4	Camera page of APP	

LIST OF ABBREVIATIONS

ABBREVIATION	DEFINITION
CNN	Convolutional Neural Network
GUI	Graphical User Interface
API	Application Programming Interfaces

CHAPTER 1
INTRODUCTION

1.1 ABOUT THE PROJECT

Sign Language Recognition System is a solution designed to help individuals who are deaf or hard of hearing communicate more effectively with those who do not understand sign language. This mobile app utilizes advanced technologies in computer vision, deep learning, and natural language processing to translate sign language gestures into spoken or written language in real time.

The goal of this project is to create an app that recognizes and translates sign language gestures accurately and instantly. By leveraging computer vision techniques, the app analyzes hand movements, facial expressions, and body postures associated with sign language, allowing for precise recognition of gestures. A powerful deep learning model called DenseNet121 is trained on a large dataset of sign language gestures to enhance the accuracy of the recognition process.

Once the gestures are recognized, the app uses natural language processing to convert them into spoken language or written text in real time. This enables individuals who do not understand sign language to comprehend and respond to the gestures, promoting better communication and inclusivity.

The user interface of the mobile app is designed to be user-friendly and accessible. Users can easily capture sign language gestures, view real-time translations, and customize the app according to their preferences. Data security measures are also implemented to protect user information and ensure privacy.

By developing the Sign Language Translation System, this project aims to empower individuals with hearing impairments to communicate more effectively with others. The app provides a convenient and immediate translation of sign language gestures, bridging the communication gap and promoting inclusivity in various social and professional settings.

1.2 PLAN OF THE REPORT

The report organized into 5 chapters. After this introductory chapter, the project report is

organized as follows:

Chapter 2: *Problem definition and Feasibility analysis*: Problem definition describes about existing system and proposed system. The feature section lists all the features and capabilities that are included in the system. It also contains feasibility study of the system which had been done before starting the project. Feasibility study provides a summary about technical feasibility, operational feasibility and economic feasibility of the system.

Chapter 3: *Software requirement and specification*: It gives the key points on the requirement specification in developing the proposed system. It is divided into 3 parts such as introduction which describes purpose and scope of the system, system specification which specify about software and hardware requirements and finally the user requirements which includes functional, non- functional and performance requirement of the system.

Chapter 4: *System design*: It provides details about system design such as use case diagram and data flow diagram. It also gives the description about modules present in the proposed system. The various tools and technologies used for the implementation has been discussed.

Chapter 5: *Coding, Testing and Implementation*: It deals about how the coding and implementation of the system has been done. It also provides a summary on the testing process that has been done after implementation. It deals with unit testing, validation testing, functional testing and GUI testing. Some test cases are also given.

Chapter 6: *Conclusion and future work*: It concludes the report and gives some insight on future enhancement.

CHAPTER 2
PROBLEM DEFINITION & FEASIBILITY ANALYSIS

2.1 INTRODUCTION

The Problem Definition and Feasibility Analysis phase is a critical early stage in the development of a knowledge hub. This phase helps to ensure that the project is well-defined and feasible, and that the knowledge hub will effectively meet the needs of its users.

2.2 PROBLEM DEFINITION

The problem addressed by the existing system is the communication barrier faced by individuals who are deaf or hard of hearing when interacting with individuals who do not understand sign language. The current methods of manual interpretation or written communication are often limited in availability, accessibility, and real-time translation, leading to challenges in effective and inclusive communication.

Specifically, the problem can be defined as follows:

The existing system lacks a real-time sign language translation solution that accurately and immediately converts sign language gestures into spoken or written language. This results in a communication gap between sign language users and non-sign language users, hindering effective communication and inclusivity in various social and professional settings.

By addressing this problem, the proposed system aims to develop a mobile app that utilizes computer vision, deep learning, and natural language processing to provide real-time and accurate translation of sign language gestures. The app will bridge the communication gap and enable individuals with hearing impairments to communicate more effectively and participate fully in conversations and interactions.

2.3 EXISTING SYSTEM

The existing system for sign language communication primarily relies on manual interpretation or the use of sign language interpreters to facilitate communication between individuals who are deaf or hard of hearing and those who do not understand sign language. However, this system has several limitations that hinder effective and inclusive communication. These limitations include:

1. Limited availability of sign language interpreters: Qualified sign language interpreters may not be readily available in all situations, leading to communication barriers for individuals who rely on sign language. The availability of interpreters may be limited, especially in informal or spontaneous interactions.

2. Cost and scheduling constraints: Hiring a sign language interpreter can be expensive, and scheduling their services in advance may not always be feasible. The cost and time required to arrange for an interpreter can result in delays or limited access to interpretation services, particularly for individuals with limited financial resources.

3. Communication barriers in remote areas: In remote or rural areas, access to sign language interpretation services may be even more limited. The lack of available interpreters and infrastructure for communication technologies can hinder effective communication for individuals who are deaf or hard of hearing, especially in areas with limited resources.

4. Potential for misinterpretation: Manual interpretation is subjective and can be prone to errors or misinterpretation. The interpreter may not always accurately capture the intended meaning of sign language gestures, leading to misunderstandings or miscommunication between the parties involved.

5. Lack of real-time translation: The existing system often lacks real-time translation capabilities, relying on the involvement of an interpreter or written communication. This can introduce delays in the communication process, hindering the fluidity and spontaneity of conversations and interactions.

6. Limited expressiveness: Written communication, while accessible in some cases, lacks the richness and expressiveness of sign language. Sign language relies on visual cues, facial expressions, and body language, which may not be effectively conveyed through written text alone, limiting the expressiveness and nuances of the communication.

7. Dependence on literacy skills: Written communication assumes that both parties possess adequate reading and writing skills. This can be a barrier for individuals with limited literacy skills or for those who are not proficient in the language used for written communication, reducing accessibility and inclusivity.

These limitations highlight the need for an improved system that offers real-time and accurate translation of sign language gestures, enhances accessibility, and overcomes the challenges faced by individuals who are deaf or hard of hearing in effective communication.

2.4 PROPOSED SYSTEM

The proposed system, the Sign Language Recognition System, aims to overcome the limitations of the existing system by providing real-time recognition and translation of sign language gestures. The system utilizes advanced technologies, such as computer vision, deep learning, and natural language processing, to accurately recognize and interpret sign language gestures in real time.

The key components of the Sign Language Recognition System include data collection and preprocessing, model training and optimization, and the development of a mobile app for sign language recognition.

In the data collection and preprocessing phase, sign language gesture data is captured using the device's camera. The captured video frames are processed using computer vision techniques to extract relevant features, such as hand movements, facial expressions, and body postures associated with sign language gestures. The preprocessed data is then prepared for further analysis.

The model training and optimization phase involves training a deep learning model, such as DenseNet121, using the collected sign language gesture data. Various techniques, including data augmentation, regularization, and hyperparameter tuning, are applied to improve the model's accuracy and performance. The trained model is then exported into a format suitable for deployment on mobile devices.

For the development of the mobile app, the Sign Language Recognition System utilizes the Flutter framework with Dart programming language. The app integrates the trained deep learning model and provides real-time sign language recognition functionality. Users can capture sign language gestures using the device's camera, and the app processes the captured video frames to recognize the gestures. The recognized gestures are then displayed or translated into spoken language or written text in real time.

The user interface of the mobile app is designed to be user-friendly and accessible. Users can easily capture sign language gestures, view real-time recognition results, and customize app settings according to their preferences. Accessibility features, such as adjustable font sizes, color schemes, and user-friendly controls, are incorporated to ensure inclusivity and accommodate different user needs.

By implementing the Sign Language Recognition System, individuals who are deaf or hard of hearing can communicate more effectively with others who do not understand sign language. The system provides a convenient and immediate translation of sign language gestures, bridging the communication gap and promoting inclusivity in various social and professional settings.

2.4.1 FEATURES

The Sign Language Recognition System will include the following key features:

1. Sign Language Recognition: The app will utilize computer vision techniques to accurately recognize and interpret sign language gestures captured through the device's camera in real-time. This involves detecting and tracking hand movements, facial expressions, and body postures associated with sign language.
2. Deep Learning Model: A powerful Convolutional Neural Network (CNN) model, such as DenseNet121, will be trained using a large dataset of sign language gestures. This model will learn the patterns and variations in different signs, enabling accurate recognition and classification of gestures.
3. Translation into Spoken or Written Language: The recognized sign language gestures will be translated into spoken language using speech synthesis technology or converted into written text. This will allow individuals who do not understand sign language to comprehend and respond to the gestures effectively.
4. Real-Time Communication: The app will provide a seamless real-time communication experience, enabling individuals using sign language to engage in conversations with non-sign language users. The translated output will be displayed or spoken in real-time, facilitating efficient and effective communication.
5. User-Friendly Interface: The mobile app will feature an intuitive and user-friendly interface, designed with a focus on accessibility. Users will be able to easily capture sign language gestures, view translations, adjust settings, and customize the app to suit their individual needs.
6. Learning Resources: The app may include learning resources such as tutorials, videos, or interactive lessons to help users improve their sign language skills. This feature promotes continuous learning and encourages users to expand their sign language

proficiency.

7. Cross-Platform Compatibility: The mobile app is developed using Flutter and Dart, providing cross-platform compatibility for both iOS and Android devices. This ensures that a wider range of users can access and benefit from the app regardless of their preferred mobile platform.

2.5 FEASIBILITY ANALYSIS

A feasibility study is made to see if the project on completion will serve the purpose of the people for the amount of work, effort and time that is spent on it. Feasibility study is a test of system proposed regarding its workability, impact on the people or organization, ability to meet the needs and effective use of resources. Feasibility study for the proposed system has made on the three categories known as

- Technical Feasibility
- Operational Feasibility
- Economic Feasibility

Here is a feasibility analysis for a sign language recognition system:

2.5.1 OPERATIONAL FEASIBILITY

Operational feasibility assesses the practicality and effectiveness of implementing and operating the Sign Language Recognition System within the intended environment. It focuses on evaluating the availability of necessary resources, the ease of system use, and the support systems required for successful deployment. Here are some key considerations for operational feasibility:

- Resource Availability: Evaluate if you have the necessary skilled personnel and resources, such as developers and hardware, to develop and maintain the system.
- Infrastructure: Check if the existing network connectivity, storage capacity, and computing power are sufficient to support real-time processing of sign language gestures.
- User Interface Design: Ensure that the user interface is user-friendly, intuitive, and accessible, with adjustable font sizes and clear visual feedback.
- Training and Support: Provide adequate training materials and establish a support

system to assist users and address their queries or technical issues.

- Integration with Existing Systems: Assess compatibility with existing systems or platforms used by individuals with hearing impairments to ensure seamless integration.
- Scalability and Flexibility: Design the system to accommodate potential growth and future enhancements, considering factors such as data storage and the ability to add new sign language gestures or dialects.

By considering these factors, the Sign Language Recognition System has found operationaly feasible

2.5.2 TECHNICAL FEASIBILITY

Technical feasibility refers to assessing whether the proposed Sign Language Recognition System can be developed and implemented using the available technologies and resources. Here are some key considerations for technical feasibility:

- Hardware and Software: Check if the necessary hardware, such as mobile devices with suitable camera capabilities, and software components are available to support the system's requirements.
- Development Expertise: Evaluate if there are skilled developers with expertise in computer vision, deep learning, and mobile app development. Determine if additional training or hiring is needed to successfully build the system.
- Compatibility: Ensure that the system can be developed for different mobile platforms, such as iOS and Android, and that the selected development framework supports cross-platform development.
- Data Collection and Processing: Assess the feasibility of collecting sufficient sign language gesture data for training the model. Determine if suitable datasets are available or if resources can be allocated for data collection and labeling.
- Model Training and Optimization: Check if the computational resources and training techniques are available to train and optimize the deep learning model for high accuracy and real-time performance.
- Deployment and Maintenance: Evaluate the feasibility of deploying the system on mobile devices and maintaining its stability and performance over time. Consider the

necessary infrastructure and ongoing maintenance efforts.

By considering the factors listed above, the system has found technically feasible.

2.5.3 ECONOMIC FEASIBILITY

Cost of Development: The project requires investments in development resources, including skilled software developers, designers, and domain experts. The availability of the required budget for development and ongoing maintenance should be assessed.

Revenue Generation: Consideration should be given to potential revenue streams, such as app sales, in-app purchases, or subscription models, to ensure the project's economic viability. So, the system has found economically feasible.

CHAPTER 3
SOFTWARE REQUIREMENT SPECIFICATION

3.1 INTRODUCTION

The Software Requirements Specification serves as a crucial initial step in software development, as it outlines the functional and non-functional requirements of the software product to be developed. This chapter provides a comprehensive description of the software system's intended behavior, constraints, and objectives. By defining the requirements of the software, the chapter establishes a clear understanding of what needs to be built and helps ensure that the software product meets the needs and expectations of its users.

3.1.1 PURPOSE

The purpose of the Sign Language Recognition System is to facilitate effective communication between individuals who use sign language and those who do not understand sign language. The system aims to overcome the communication barrier faced by individuals with hearing impairments by providing real-time recognition and translation of sign language gestures into spoken or written language.

The system's purpose is to empower individuals with hearing impairments to express themselves and communicate more effectively in various social and professional interactions. By accurately recognizing and translating sign language gestures, the system promotes inclusivity, equality, and accessibility in communication.

Additionally, the Sign Language Recognition System aims to increase the independence and autonomy of individuals with hearing impairments. By providing a mobile app that can be readily accessed and used on common smartphones, the system enables users to communicate in sign language without relying solely on sign language interpreters or manual interpretation.

The purpose of the Sign Language Recognition System is to bridge the communication gap between individuals who use sign language and those who do not, ultimately fostering better understanding, inclusivity, and equal participation in society.

3.1.2 SCOPE

The scope of the Sign Language Recognition System encompasses the development and

implementation of a mobile application that provides real-time recognition and translation of sign language gestures. The system is designed to be compatible with both iOS and Android platforms, ensuring widespread accessibility and usability.

The system's primary focus is on accurately recognizing and interpreting sign language gestures captured through the device's camera. It leverages computer vision techniques, deep learning models, and natural language processing algorithms to achieve high accuracy and real-time performance.

The application's user interface is designed to be user-friendly and intuitive, allowing users to easily capture sign language gestures and view the corresponding translations in spoken language or written text. The system may also include customization options, such as adjustable font sizes and color schemes, to cater to individual user preferences and accessibility needs.

The Sign Language Recognition System aims to support a wide range of sign language gestures, including hand movements, facial expressions, and body postures, to facilitate comprehensive and expressive communication. It should be capable of recognizing a diverse set of gestures commonly used in sign language.

The system's scope may also include considerations for data privacy and security, ensuring that user information and interactions are protected. It may incorporate measures such as data encryption, secure storage, and user authentication to safeguard sensitive information.

It's important to note that the system's scope may vary depending on project requirements, available resources, and technical constraints. The primary goal is to provide an efficient, reliable, and accessible solution for sign language recognition and translation, ultimately promoting effective communication and inclusivity for individuals with hearing impairments.

3.2 SYSTEM REQUIREMENTS

3.2.1 HARDWARE SPECIFICATION

- Processor: Intel core i3 or above
- Clock Speed: 500MHZ
- System Bus: 32Bit or above
- RAM: Minimum 4GB (recommended8GB)
- HDD: 20GB

3.2.2 SOFTWARE SPECIFICATION

- Operating System: Windows 11/10/08 or Linux
- Frontend: Dart, Flutter
- Backend: Python, Flask
- Browser: Mozilla Firefox/ Google Chrome etc.
- Connection: TCP / IP
- Protocol: HTTP, SMTP.

3.3 USER REQUIREMENTS

User requirements refer to the specific needs and expectations of the end-users of a software system. These requirements are critical to the success of the system, as they directly impact the user's experience and satisfaction with the software.

3.3.1 FUNCTIONAL REQUIREMENT

- Sign Language Gesture Recognition: The system should accurately recognize and classify a wide range of sign language gestures, including hand movements, facial expressions, and body postures.
- Real-Time Translation: The system should provide real-time translation of recognized sign language gestures into spoken language or written text.
- Gesture Customization: The system should allow users to customize or add new sign language gestures based on their individual preferences or specific sign language dialects.
- User Interface: The user interface should be intuitive, user-friendly, and accessible, allowing users to easily capture sign language gestures and view translations.
- Gesture Storage: The system should provide a mechanism to store and manage a library of sign language gestures for efficient recognition and translation.
- Integration with Device Camera: The system should utilize the device's camera to capture sign language gestures for recognition.
- Language Selection: The system should support multiple spoken languages for

translation, allowing users to choose their desired language for communication.

These functional requirements outline the core features and capabilities of the Sign Language Recognition System, ensuring that it can accurately recognize and translate sign language gestures in real time, provide customization options, and offer a user-friendly interface for seamless communication.

3.3.2 NON-FUNCTIONAL REQUIREMENT

- Usability: The system should have a clear and intuitive user interface, making it easy for users to navigate and interact with the application.
- Reliability: The system should be reliable and perform consistently in different environments and lighting conditions.
- Accessibility: The system should consider accessibility features such as adjustable font sizes, color schemes, and user-friendly controls to cater to diverse user needs.
- Security: The system should implement appropriate security measures to protect user data and ensure the privacy and confidentiality of user interactions.
- Compatibility: The system should be compatible with a range of mobile devices, operating systems (iOS and Android), and screen resolutions.
- Scalability: The system should be scalable to handle increased user demand and a growing library of sign language gestures without significant performance degradation.

These non-functional requirements ensure that the Sign Language Recognition System not only provides accurate recognition and translation but also prioritizes usability, reliability, security, compatibility, performance, and resilience. They contribute to an overall positive user experience and the system's ability to meet the needs and expectations of its users.

3.3.3 PERFORMANCE REQUIREMENT

- Real-Time Recognition: The system should have fast and accurate recognition capabilities, providing near-instantaneous translation of sign language gestures into spoken or written language.
- Low Latency: The system should minimize the delay between capturing a sign language

gesture and displaying the corresponding translation.

- Accuracy: The system should achieve a high level of accuracy in recognizing and classifying sign language gestures, minimizing misinterpretation and errors.
- Responsiveness: The system should respond quickly to user interactions, providing immediate feedback and translations.
- Efficiency: The system should optimize resource usage, such as CPU and memory, to ensure efficient performance without significant battery drain on the mobile device.
- Robustness: The system should be able to handle variations in lighting conditions, different hand sizes, and subtle differences in sign language gestures, maintaining accurate recognition performance.

These performance requirements ensure that the Sign Language Recognition System delivers accurate and real-time recognition and translation of sign language gestures while optimizing resource usage, scalability, and adaptability to provide a seamless and efficient user experience.

3.4 EXPLANATION OF TECHNOLOGIES USED

The Sign Language Recognition System utilizes several advanced technologies to achieve its objectives. Here is an explanation of the key technologies used in the project:

3.4.1. Flutter and Dart:

Flutter is a cross-platform UI framework developed by Google, and Dart is the programming language used to write Flutter applications. Flutter allows for the development of mobile apps that can run on both iOS and Android platforms, providing a seamless user experience and reducing development time and effort.

3.4.2. Convolutional Neural Networks (CNN):

CNN is a deep learning architecture commonly used in computer vision tasks. In the proposed system, a DenseNet121 model, which is a type of CNN, is employed for sign language gesture recognition. CNNs excel at extracting meaningful features from images, enabling accurate recognition and interpretation of sign language gestures.

3.4.3. MediaPipe Holistic:

MediaPipe Holistic is a computer vision model provided by Google that offers multi-modal perception, including hand tracking, pose estimation, and face detection. This model is used to

extract key points related to hand movements, facial expressions, and body postures from the captured sign language gesture images.

3.4.4. TensorFlow and TensorFlow Lite:

TensorFlow is an open-source machine learning framework developed by Google. In the proposed system, TensorFlow is utilized for model training and optimization. TensorFlow Lite, a lightweight version of TensorFlow, is used to export and deploy the trained DenseNet121 model on mobile devices. TensorFlow Lite enables efficient inference on resource-constrained platforms like smartphones.

3.4.5. Data Augmentation:

Data augmentation is a technique used to increase the diversity of training data by applying various transformations to the existing dataset. In the proposed system, data augmentation is employed to generate augmented images with variations in lighting, rotation, scaling, and other parameters. This helps improve the model's generalization and robustness.

3.4.6. Deep Learning Optimization Techniques:

Several optimization techniques are applied during model training to enhance performance and prevent overfitting. These techniques include regularization methods (e.g., dropout, L1/L2 regularization), batch normalization, and learning rate scheduling. They contribute to better model accuracy and stability.

3.4.7. Natural Language Processing (NLP):

NLP techniques are utilized to translate recognized sign language gestures into spoken or written language. NLP algorithms process the output from gesture recognition and convert it into text or speech, ensuring effective communication between sign language users and non-sign language users.

3.4.8. Flask API:

Flask API is a web framework for building APIs (Application Programming Interfaces) in Python. It allows developers to create and deploy web services that can be accessed by other applications or clients over the internet. Flask API simplifies the process of creating APIs by providing a lightweight and flexible framework with a minimalistic design. Flask API provides

a flexible and efficient framework for building robust and scalable APIs in Python. It is widely used in web development to create RESTful APIs, microservices, and backend services that can be easily consumed by other applications or clients.

3.5 Algorithm Definition

3.5.1. Convolutional Neural Network:

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning system that can take an input picture and assign importance (learnable weights and biases) to various aspects/objects in the image, as well as differentiate between them. The amount of pre-processing required by a ConvNet is much less than that required by other classification techniques. ConvNets can learn these filters/characteristics with adequate training, whereas simple techniques need hand-engineering of filters. ConvNets are multilayer artificial neural networks designed to handle 2D or 3D data as input. Every layer in the network is made up of several planes that may be 2D or 3D, and each plane is made up of numerous independent neurons composition, where nearby layer neurons are linked but same layer neurons are not.A ConvNet can capture the Spatial and Temporal aspects of an image by applying appropriate filters. Furthermore, reducing the number of parameters involved and reusing weights resulted in the architecture performing better fitting to the picture collection. ConvNet's major goal is to make image processing easier by extracting relevant characteristics from images while preserving crucial information that is must for making accurate predictions. This is highly useful for developing an architecture that is not just capable of collecting and learning characteristics but also capable of handling massive volumes of data.

- Overall Architecture:

CNNs are made up of three different sorts of layers. There are three types of layers: convolutional layers, pooling layers, and fully-connected layers. A CNN architecture is generated when these layers are layered. Figure 1 depicts a simple CNN architecture for Sign classification.

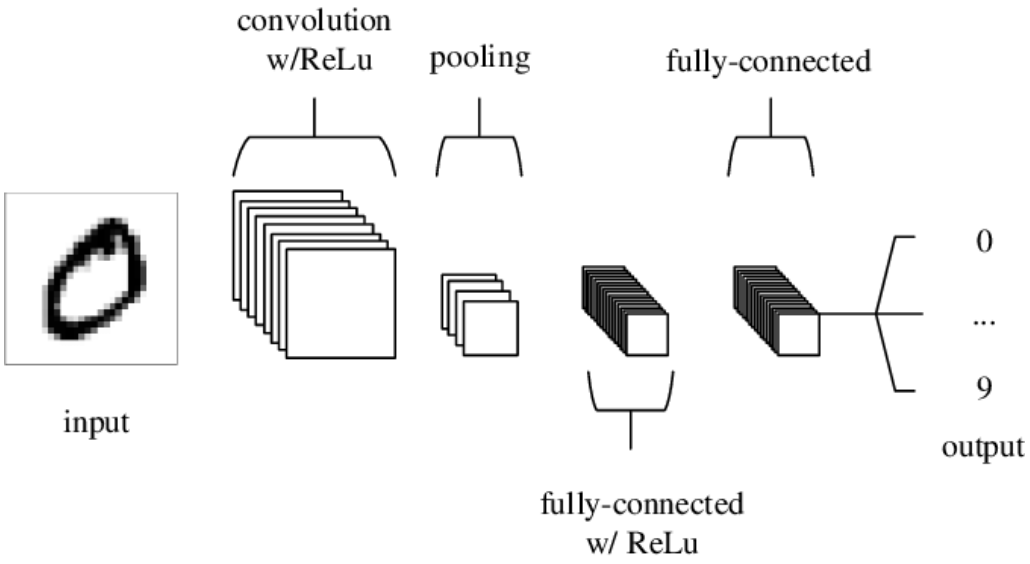

Fig 3.1 CNN Architecture

- Basic Layers in Convolutional Neural Networks

  - Convolutional layers

The main purpose of a convolutional layer is to detect features or visual features in images such as edges, lines, color drops, etc. It utilized various filters (also known as kernels, feature detectors), to detect features are present throughout an image. A filter is just a matrix of values, called weights, that are trained to detect specific features. The filter carries out a convolution operation, which is an element-wise product and sum between two matrices. The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. CNNs can include one or multiple Convolutional Layer. Where, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well.

The output matrix from the convolution layer called feature maps or convolved features. Since we have multiple filters, we end up with a 3D output: one 2D feature map per filter. The filter must have the same number of channels as the input image so that the element-wise multiplication can take place.

Conv layer main Arguments

1.kernel_size:

filter dimensions

2.strides:

The stride value dictates by how much the filter should move at each step.

3.Padding:

Typically, convolution results in a bit small image than the input one. One solution to resolve it, pad the image with zeros(zero-padding) to allow for more space for the kernel to cover the image. Adding padding to an image processed by a CNN allows for a more accurate analysis of images.

4.Activation Function:

The feature maps are summed with a bias term and passed through a non-linear activation function. The purpose of the activation function is to introduce non-linearity into our network because the images are made of different objects that are not linear to each other, so the images are highly non-linear. Rectified Linear Unit(ReLU) is mostly preferred because it provides sparsity and a reduces likelihood of vanishing gradient problems.

- Pooling Layer

Pooling Layer down samples the feature maps (to save on processing time), while also reducing the size of the image. This helps reduce overfitting, which would occur if CNN is given too much information, especially if that information is not relevant in classifying the image. There are different types of pooling, for example, max pooling and min pooling Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel. Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that Max Pooling performs a lot better than Average Pooling.

These values then form a new matrix called a pooled feature map.

- Flattening Layer

After extracting features using multiple convolution and pooling layers, we are going

to flatten the final output and feed it to a regular Neural Network for classification purposes. Flattening layer convert the 3D representation into a long feature vector.

- Dense layers

  The flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training. Sigmod and Softmax are added in the last dense layer for binary and multiclass classification respectively.

- Global Pooling Layers

  You can use another strategy called global pooling to replace the Flatten layers in CNN. It generates one feature map for each corresponding category of the classification task in the last Conv layer. One advantage of global pooling over the fully connected layers is that it is more native to the convolution structure by enforcing correspondences between feature maps and categories. Another advantage is that there is no parameter to optimize in the global pooling thus overfitting is avoided at this layer. It also can be used to reduce the dimensionality of the feature maps.

  There are 2 types of Global pooling:

  Global Average Pooling: which can be viewed as a structural regularizer that explicitly enforces feature maps to be confidence maps of concepts (categories).

  Global Max Pooling: downsamples the input representation by taking the maximum value over the time dimension.

- Batch normalization

  Batch normalization is a layer that allows every layer of the network to do learning more independently. It is used to normalize the output of the previous layers to avoid having instability in network and having high weight cascading down to the output. The activations scale the input layer in normalization. Using batch normalization learning becomes efficient also it can be used as regularization to avoid overfitting of the model. It can be used at several points in between the layers of the model. It is often placed after the convolution and pooling layers.

- Dropout

  Dropouts are the regularization technique that is used to prevent overfitting in the model. Dropouts randomly drop some percentage of neurons of the network. This is done to enhance the learning of the model. Dropouts are usually advised not to use after the convolution layers, they are mostly used after the dense layers of the network.

3.5.2. DenseNet 121

Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. In this paper, we embrace this observation and introduce the Dense Convolutional Network (DenseNet), which connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections - one between each layer and its subsequent layer - our network has L(L+1)/2 direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. We evaluate our proposed architecture on four highly competitive object recognition benchmark tasks (CIFAR-10, CIFAR-100, SVHN, and ImageNet). DenseNets obtain significant improvements over the state-of-the-art on most of them, whilst requiring less memory and computation to achieve high performance.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv, 3 × 3 conv] × 24	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48	[1 × 1 conv, 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv, 3 × 3 conv] × 16	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Fig 3.2 DenseNet Architectures

3.5.2. FLASK API

In the proposed Sign Language Recognition System, the Flask framework is used to develop an API (Application Programming Interface) that serves as the backend for the system. Flask is a lightweight and flexible web framework for Python that allows developers to build web

applications and APIs.

The Flask API acts as a bridge between the frontend of the system (such as the mobile app) and the backend processes responsible for sign language recognition and translation. It handles incoming requests from the frontend, processes the data, and sends back the appropriate response.

Here's how Flask API can be integrated into the system:

1. API Endpoints: Flask allows developers to define various API endpoints that correspond to different functionalities of the system. For example, there could be an endpoint to receive sign language gesture data for recognition or an endpoint to retrieve the translation of recognized gestures. These endpoints are defined as functions or methods in the Flask application.
2. Request Handling: When a request is made to an API endpoint, Flask handles the incoming request and extracts the necessary data from the request payload. The data can include images, video streams, or other relevant information related to the sign language gestures.
3. Processing and Recognition: Once the request data is received, the Flask API can invoke the necessary processes for sign language recognition and translation. This could involve using computer vision techniques, deep learning models, and NLP algorithms to analyze the gestures and generate translations.
4. Response Generation: After the recognition and translation processes are completed, the Flask API generates an appropriate response. This could include the recognized gestures, translations in spoken or written language, or any other relevant information required by the frontend. The response is typically sent back in a structured format, such as JSON.
5. Error Handling and Validation: Flask provides mechanisms for handling errors and validating the incoming requests. It allows developers to define error handlers and implement request validation logic to ensure the integrity and security of the system.

By using Flask API, the Sign Language Recognition System can provide a scalable and efficient backend for handling requests and processing sign language gestures. The Flask API acts as an intermediary between the frontend and the backend processes, facilitating communication and enabling real-time recognition and translation of sign language gestures.

CHAPTER 4
SYSTEM DESIGN

4.1 INTRODUCTION

The system design is a critical phase in software development, where the requirements identified in the Software Requirements Specification are transformed into a detailed blueprint for the software system. It describes about use case as well as data flow diagram that has been used to develop the system.

4.2 USE CASE DIAGRAM

A use case diagram is a visual representation that illustrates the interactions between actors (users or external systems) and the system itself. It helps to identify the different use cases or functionalities of the system and how they relate to the actors involved. Here are some components you can include in the use case diagram for your project:

1.Actors:

- User: The individual who interacts with the Sign Language Recognition System through the mobile app.
- External System (optional): Any external system or service that the Sign Language Recognition System may interact with, such as a database or an authentication service.

2.Use Cases:

- Capture Sign Language Gesture: The user can capture sign language gestures using the mobile app's camera.
- Recognize Gesture: The system processes the captured gesture and recognizes the corresponding sign language gesture.
- Translate Gesture: The system translates the recognized sign language gesture into spoken language or written text.
- Display Translation: The translated gesture is displayed to the user in real-time.
- Customize Settings: The user can customize various settings of the app, such as font size or language preferences.

3.Relationships:

- Association: Connects the actors with the use cases they are involved in.
- Include: Represents a relationship where one use case includes the functionality of another use case. For example, the "Translate Gesture" use case includes the "Recognize Gesture" use case since recognition is a necessary step for translation.
- Extend: Represents a relationship where one use case can be extended by another use case. For example, the "Capture Sign Language Gesture" use case can be extended by the "Customize Settings" use case, as the settings may include options related to gesture capture.

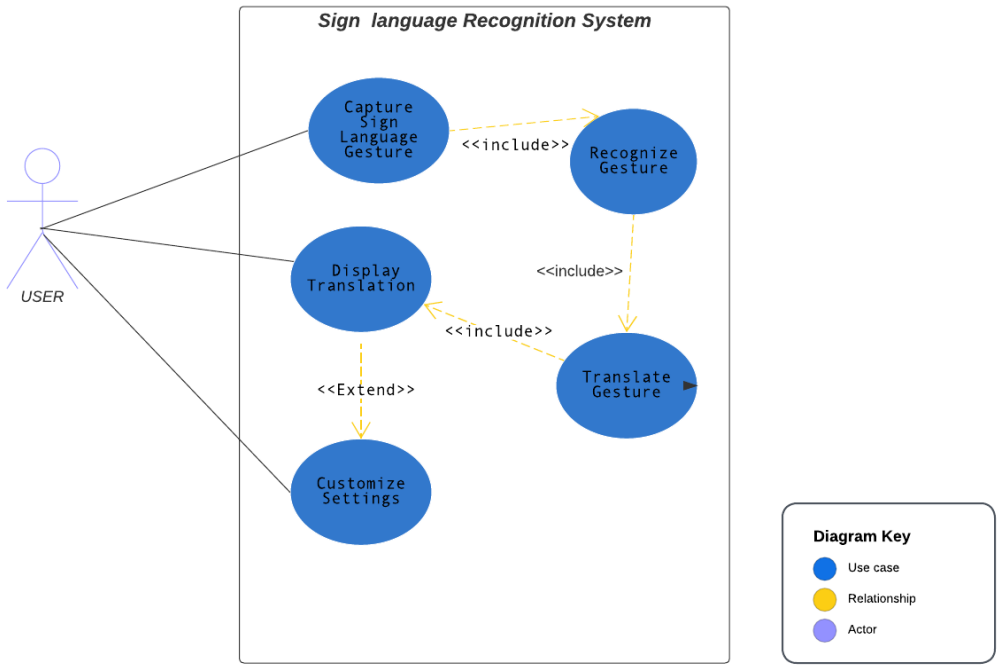

Fig 4.1 Use case diagram

4.3 DATA FLOW DIAGRAM

The proposed system consists of three main components: data collection and pre-processing, model training and optimization, and mobile app development and real-time sign language recognition.

The data collection and pre-processing component captures sign language gesture data using a webcam and extracts key points using the MediaPipe Holistic model. The pre-processing step

converts the captured images to BGR format and stores only the holistic key points with a .jpg extension.

The model training and optimization component trains a DenseNet121 model on the collected data using techniques such as data augmentation, regularization, and hyperparameter tuning. The trained model is then exported into a format such as TensorFlow Lite which can be used in the mobile app.

The mobile app development and real-time sign language recognition component develops a mobile app that integrates the exported model and allows users to capture sign language gestures using their device's camera. The app uses the trained model to recognize sign language gestures in real-time and translate them into spoken language.

This proposed system provides a high-level overview of how the different components work together to recognize sign language gestures and translate them into spoken language in real-time.

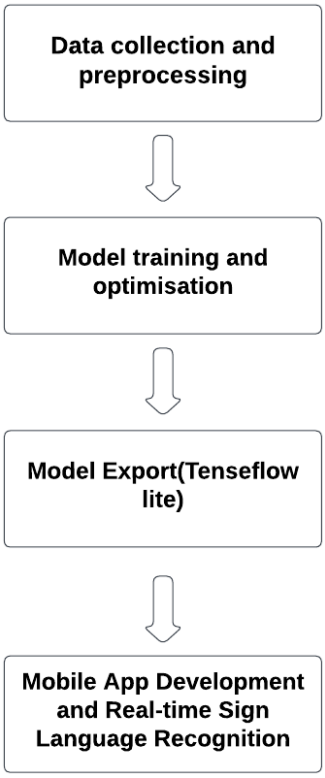

Fig 4.2 Data flow diagram

4.3.1 WORK FLOW OF DEEP LEARNING MODEL:

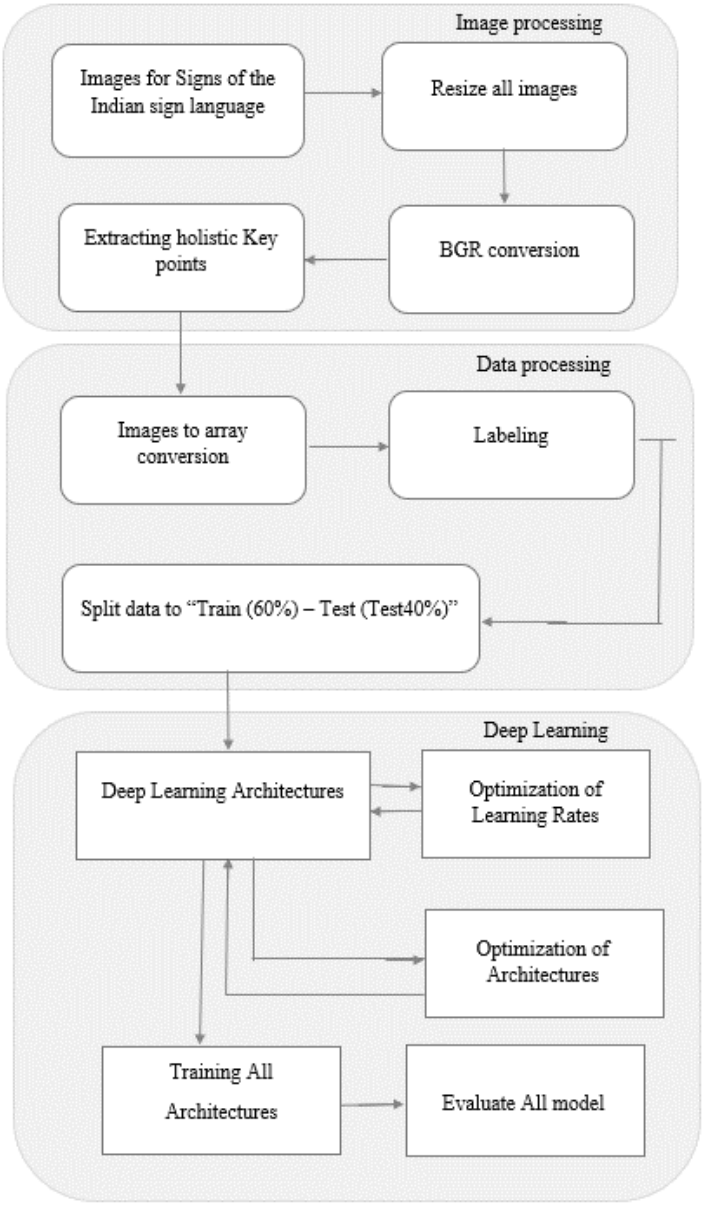

Fig 4.3 workflow of deep learning model

4.4 Class diagram:

In this diagram, the User interacts with the SignLanguageApp, which communicates with three main modules: the CameraModule, the SignDetection module, and the ModelController.

The CameraModule class is responsible for capturing the video input and providing it to other modules for processing. It has methods for starting and stopping the camera, and retrieving frames from the camera.

The SignDetection class is responsible for detecting signs in the video input. It has a detect() method that takes a frame as input and returns the detected sign. The detected sign is passed to the ModelController for translation.

The ModelController class is responsible for managing the machine learning models used in the app. It has a model attribute that is an instance of the DenseNet121 class, which is used for sign translation. It also has methods for loading the model and translating signs.

The MediaPipeHolistic class is responsible for extracting key points from the video input. It has a process_image() method that takes an image as input and returns the key points. This module is used by the SignDetection module for sign detection.

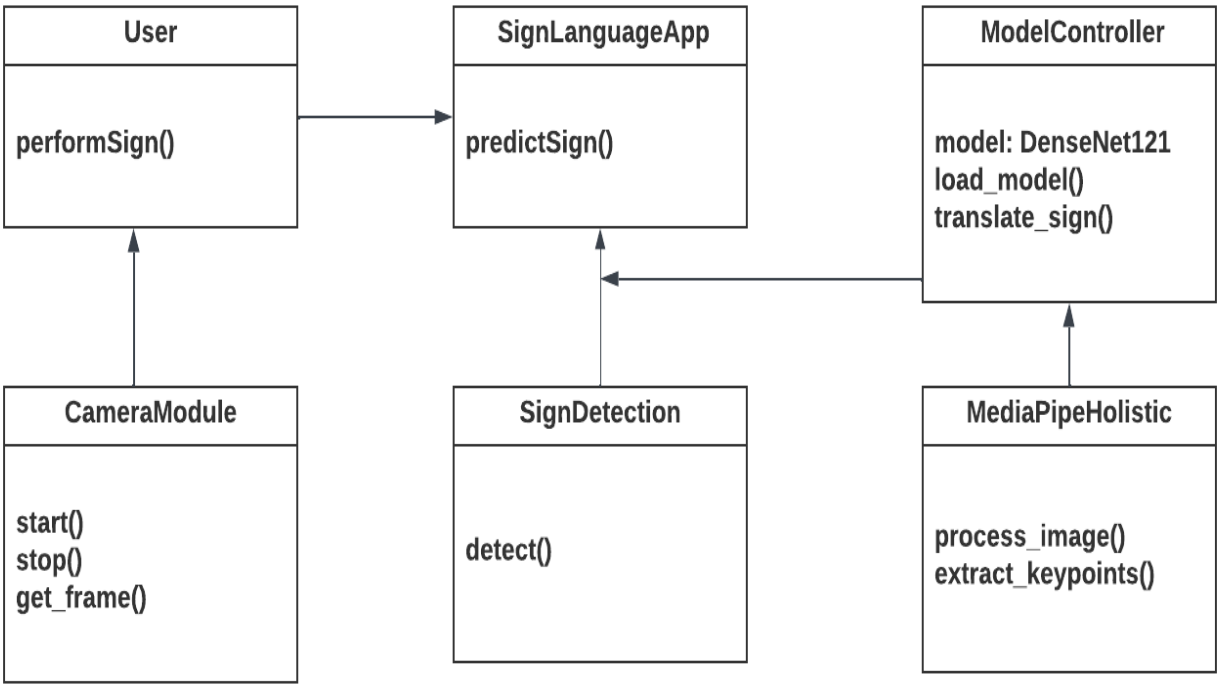

Fig 4.4 Class diagram

4.5. Sequence Diagram:

The sequence diagram shows the interactions between the main components in the proposed system: User, MobileApp, TrainedModel, and the MediaPipe API used for extracting key points from the captured images.

The sequence starts when the User performs a sign language gesture, triggering the MobileApp to capture an image from the webcam and extract key points using the MediaPipe API. The MobileApp then preprocesses the image and passes the key points to the TrainedModel for sign

recognition.

The TrainedModel predicts the sign language gesture and returns it to the MobileApp, which displays it to the User. The User can then provide feedback on the accuracy of the prediction, which is used to update the TrainedModel.

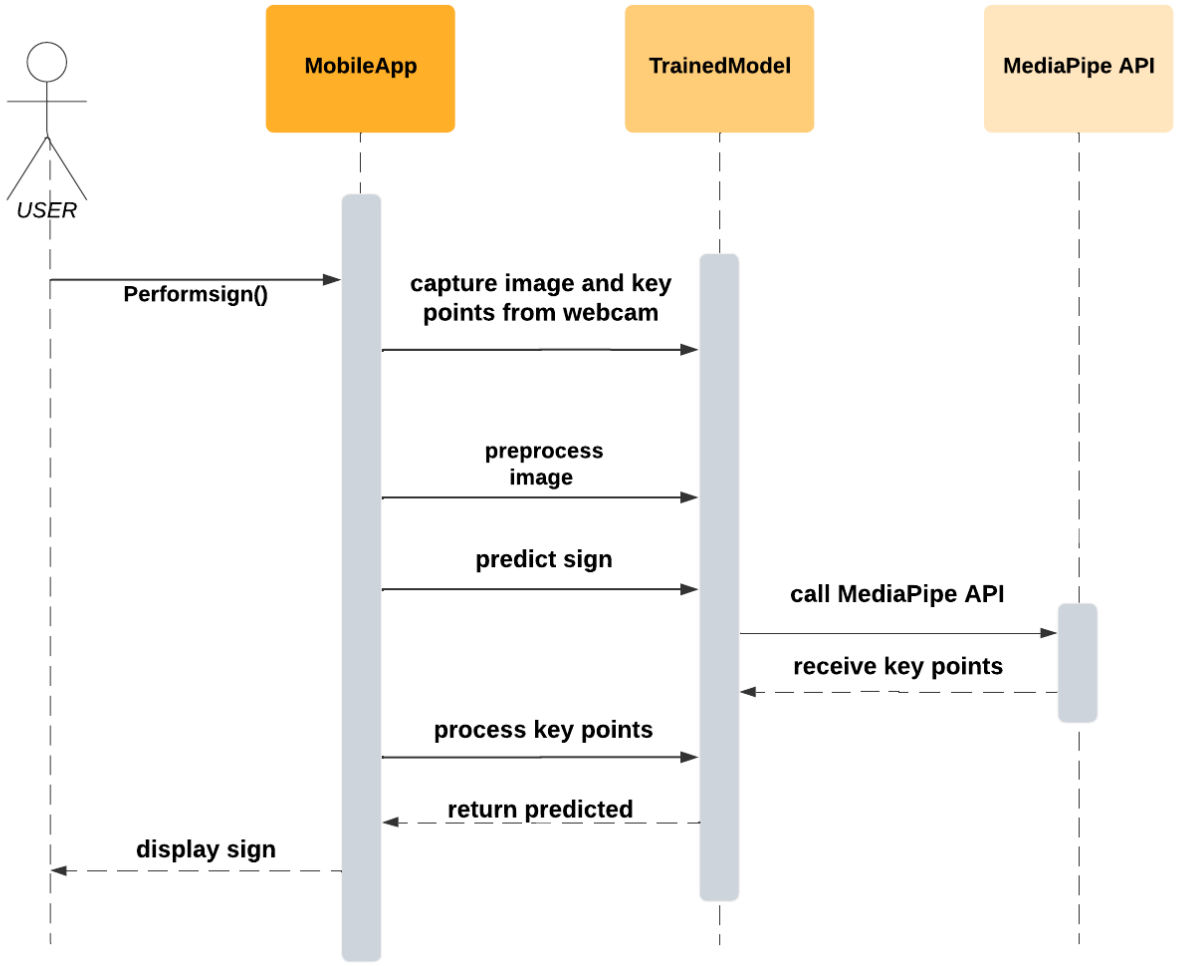

Fig 4.5 sequence diagram

CHAPTER 5
CODING, TESTING AND IMPLEMENTATION

5.1 INTRODUCTION

The coding, testing, and implementation phase of the system is a critical stage that involves the actual development of the software product. This stage involves the translation of the software design into a fully functioning software system by writing code, testing it for errors, and deploying it in the production environment.

5.2 EXPERIMENTS AND RESULTS

5.2.1 Dataset collection and Preprocessing

It is a very crucial part of the research works in all the arenas as it is fundamental to foster the development of any machine or deep learning model. However, it is full of challenges. During data collection, the biggest challenge I faced was that there were no standard datasets for Indian sign language available. Therefore, as part of this project, I attempted to manually construct a dataset that could help us overcome this problem. First of all, I captured the images using a webcam where various signs were taken into account. 42 different signs considered. The images were converted to BGR format and the key points are extracted using media pipe holistic model. The images were captured in different rotations and only the holistic key points were stored with. jpg extension.

The image is made ready for feature detection and extraction in this phase. To preserve uniformity of scale, the dimensions of all the images are kept the same. In the default option, the captured image frame is converted into RGB colour space for the images acquired with BGR. For hand segmentation, mediapipe holistic model called for both the hands and for pose landmarks. The annotates stored later converted to array form and labelled the every annotates.

Fig 5.1. Input Image of Sign A

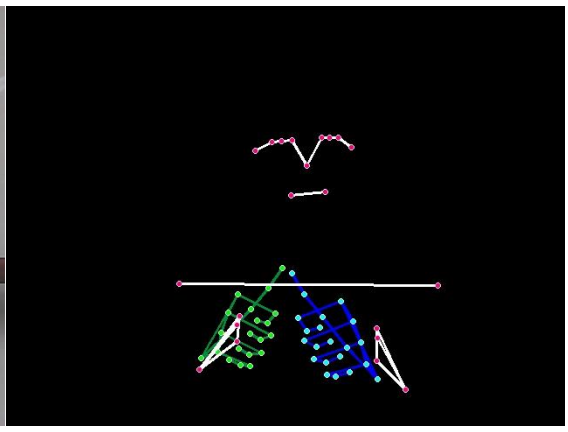

Fig 5.2 Extracted Features

5.2.2 Training and Testing

The dataset is divided into 2 sets. The training set consists of 80% of the total data and the remaining 40% is used as testing means. The models (Densenet-121) have given high accuracy on the images, Densenet-121 has performed better with a lesser no. of features. The system is trained to recognize 42 signs. Current results are promising, keeping in mind that few improvements could provide better results. Densenet-121 performance we have observed an overall accuracy of 98.651%. The total epochs are 50.

LOSS and ACCURACY

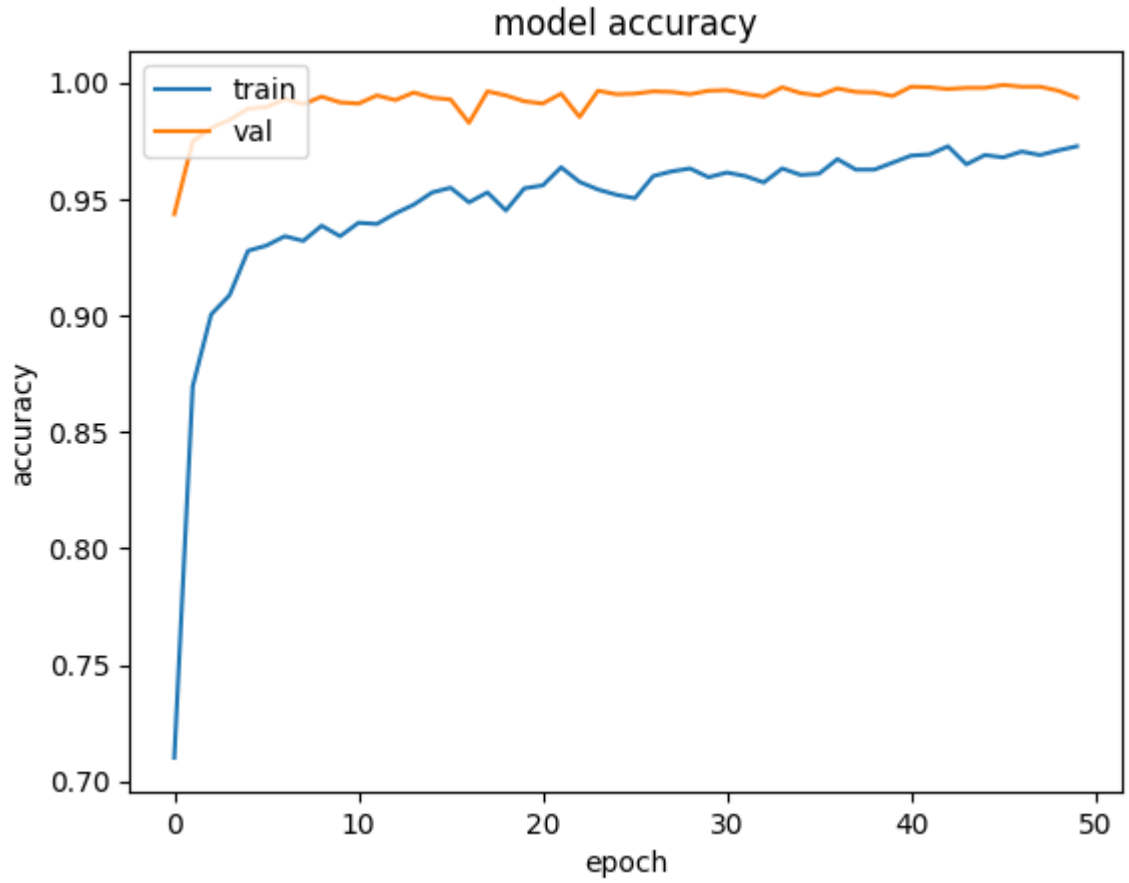

Fig 5.3 Model Accuracy Curve

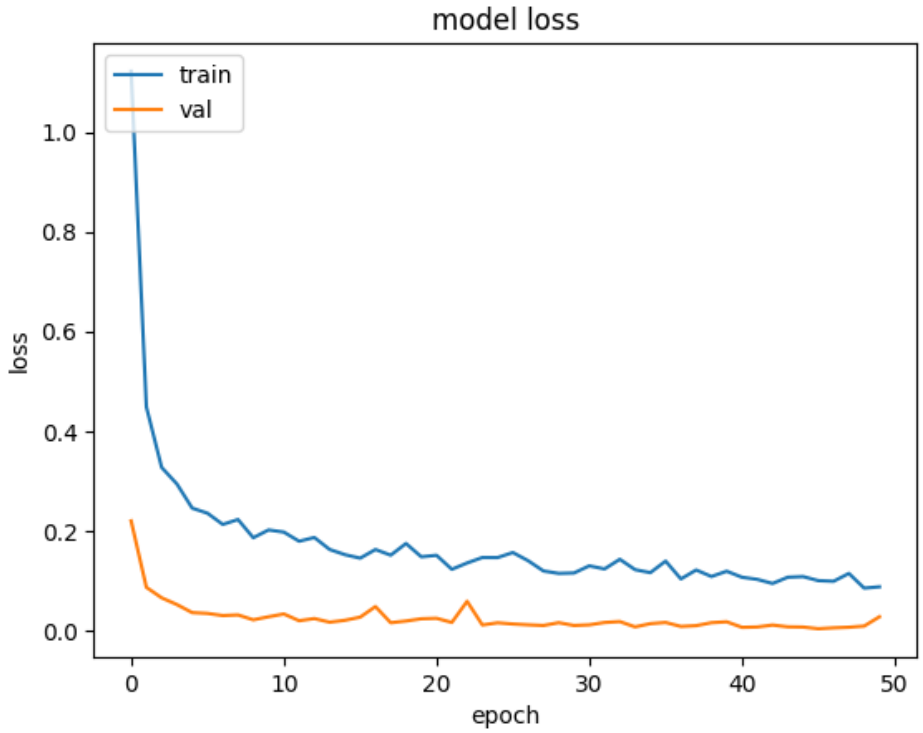

Fig 5.4 Model Lose Curve

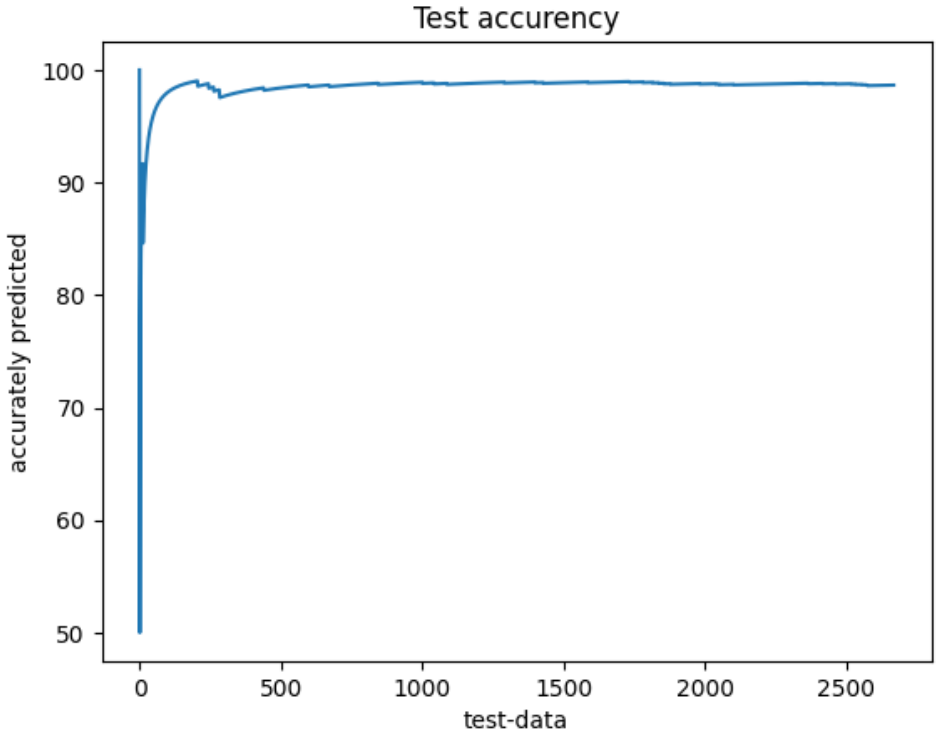

Fig 5.5 Test Accuracy Curve

PERFORMANCE METRIX – Confusion Metrix

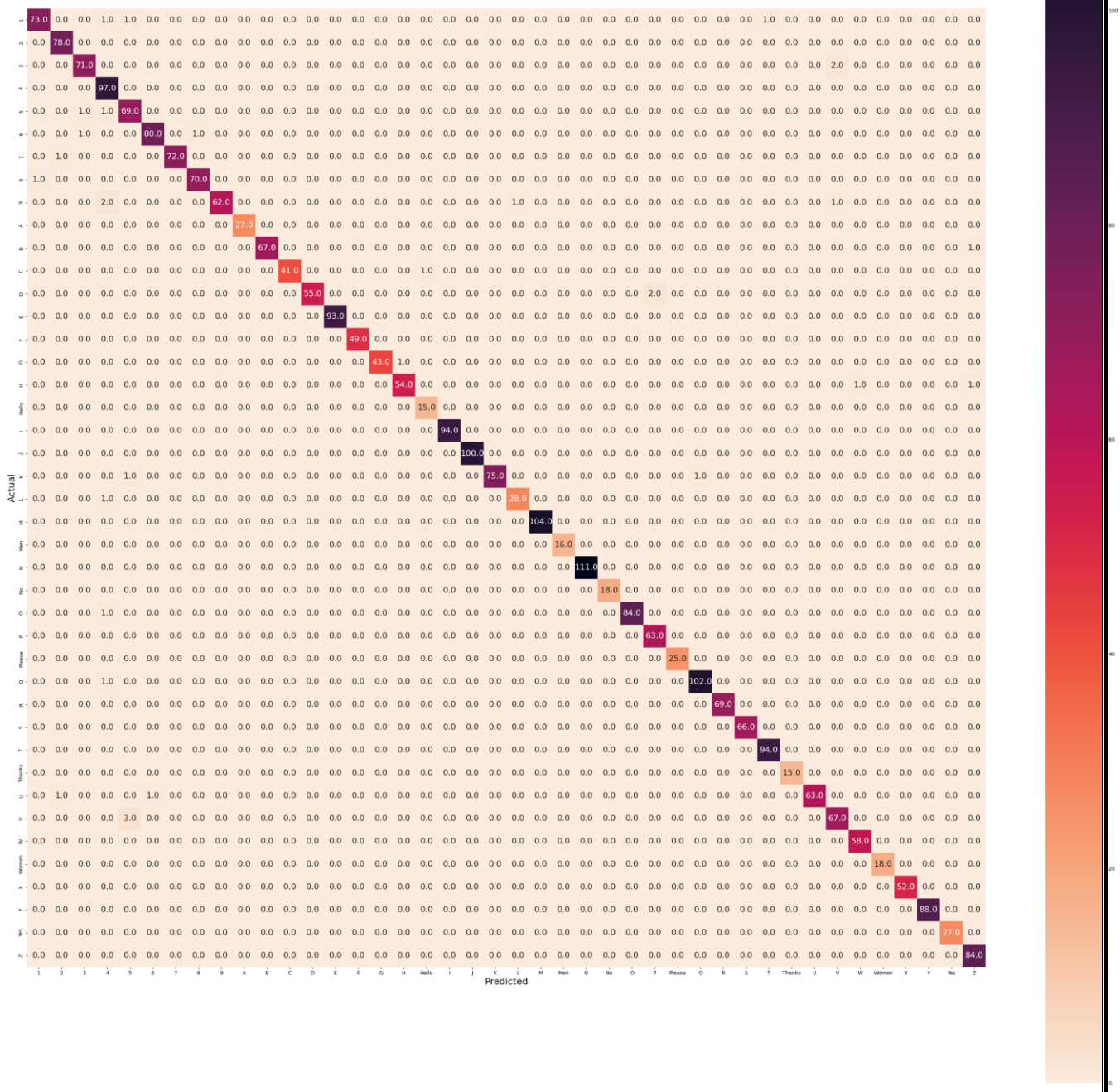

Fig 5.6 Confusion Matrix

5.2.3 API and Mobile App

The Sign Language Recognition System utilizes Flask as the backend API framework and Flutter as the frontend framework for mobile app development. Flask, a lightweight and flexible web framework in Python, is used to build the API that handles communication between the mobile app and the server. It provides the necessary endpoints for capturing sign language gestures, sending them for recognition and translation, and receiving the translated results.

On the other hand, Flutter, a cross-platform UI toolkit, is employed to develop the mobile app's frontend. Flutter offers a rich set of pre-designed widgets and libraries, making it easier to create a visually appealing and interactive user interface. The app allows users to capture sign language gestures using their device's camera, view real-time translations, and customize app settings according to their preferences.

By combining Flask and Flutter, the Sign Language Recognition System provides a seamless and efficient solution for real-time sign language translation. The Flask API ensures smooth communication between the mobile app and the server, while Flutter empowers the app with a user-friendly interface that facilitates gesture capturing, translation display, and customization. Together, these technologies enable individuals with hearing impairments to communicate effectively and bridge the communication gap with others.

5.3 IMPLEMENTATION

The implementation of the Sign Language Recognition System involves several steps to bring the project to life. Here is a brief description of the implementation process:

- Requirements Analysis: Begin by understanding and analyzing the project requirements in detail. Clarify the desired functionalities, performance expectations, and user experience.
- System Design: Design the overall system architecture, including the backend API and the frontend mobile app. Determine the data flow, component interactions, and system modules.
- Backend Development: Implement the backend API using Flask. Define the API endpoints for capturing sign language gestures, processing them for recognition, and returning the translated results. Incorporate the necessary libraries and dependencies for image processing, deep learning, and natural language processing.

- Frontend Development: Develop the mobile app using Flutter. Design the user interface with screens, buttons, and user interactions. Integrate the necessary functionalities for capturing sign language gestures, displaying real-time translations, and allowing customization options.
- Integration: Connect the frontend and backend components by establishing communication between the mobile app and the Flask API. Ensure that data is transmitted securely and efficiently between the client and server.
- Testing: Conduct thorough testing of the system to ensure its functionality, reliability, and performance. Perform unit tests, integration tests, and system tests to validate the different components and verify that they work together as expected. Identify and address any issues or bugs during the testing phase.
- Deployment: Prepare the system for deployment on the desired platforms, such as mobile devices or cloud hosting. Set up the necessary infrastructure, including servers, databases, and hosting environments. Deploy the backend API and the mobile app according to the deployment strategy.
- User Feedback and Iteration: Gather feedback from users and stakeholders to identify areas for improvement. Incorporate user feedback to refine the system and enhance its usability and effectiveness. Continuously iterate and update the system based on user needs and technological advancements.

Throughout the implementation process, ensure proper documentation, version control, and collaboration among the development team members. Regular communication and coordination are essential to ensure a smooth and successful implementation of the Sign Language Recognition System.

5.4 TESTING

Testing is a process of executing software in an intention to find errors in order to test its validity. After coding of each functionality, the system is tested for various test cases. By analyzing the performance, code is corrected and modified at necessary stages. Thus testing was done in each phase, because start of the next phase depends upon the performance of the previous stage. Each module was tested independently.

5.4.1. Unit Testing:

Unit testing is a testing technique that focuses on testing individual units or components of the software system in isolation. In the context of the Sign Language Recognition System, unit testing involves testing each module or function of the codebase independently to ensure that it behaves as expected. This helps identify and fix any issues or bugs at an early stage of development. Unit testing typically involves writing test cases that cover different scenarios and input combinations to verify the correctness of the code.

5.4.2. Validation Testing:

Validation testing, also known as user acceptance testing or customer acceptance testing, aims to evaluate the system's compliance with user requirements and ensure that it meets the intended purpose. In the case of the Sign Language Recognition System, validation testing involves engaging end-users or stakeholders to test the system and provide feedback. This testing phase focuses on verifying that the system accurately recognizes sign language gestures and translates them into the desired output, such as spoken or written language. It ensures that the system meets the needs and expectations of its intended users.

5.4.3. Functional Testing:

Functional testing is a type of testing that verifies the functional requirements of the system. It focuses on testing the system's features, functionalities, and interactions with users. In the Sign Language Recognition System, functional testing involves testing the specific functionalities such as gesture capturing, recognition accuracy, real-time translation, and customization options. The goal is to ensure that all the required functionalities are working correctly and providing the expected outputs.

5.4.4. GUI Testing:

GUI (Graphical User Interface) testing focuses on testing the user interface of the system to ensure that it is user-friendly, visually appealing, and functions correctly. In the context of the Sign Language Recognition System, GUI testing involves testing the mobile app's interface, including screens, buttons, menus, and user interactions. It verifies that the UI elements are displayed correctly, respond to user inputs appropriately, and provide the necessary feedback. GUI testing also covers accessibility aspects, ensuring that the app is accessible to individuals with hearing impairments and meets accessibility standards.

These testing techniques play a crucial role in ensuring the quality, functionality, and usability

of the Sign Language Recognition System. They help identify and address any issues or discrepancies in the system, ensuring that it performs as intended and meets the needs of its users.

5.5 CODING

- **Load the pretrained Densnet121 model**

```
model_d=DenseNet121(weights='imagenet',include_top=False, input_shape=(128, 128, 3))
x=model_d.output
x= GlobalAveragePooling2D()(x)
x= BatchNormalization()(x)
x= Dropout(0.5)(x)
x= Dense(1024,activation='relu')(x)
x= Dense(512,activation='relu')(x)
x= BatchNormalization()(x)
x= Dropout(0.5)(x)
preds=Dense(42,activation='softmax')(x)
```

- **Preparing the training data**

```
from google.colab import drive
drive.mount('/content/drive')
data=[]
labels=[]
random.seed(42)
imagePaths = sorted(list(os.listdir("/content/drive/MyDrive/collected_images//")))
#print(imagePaths)
random.shuffle(imagePaths)
for img in imagePaths:
    path=sorted(list(os.listdir("/content/drive/MyDrive/collected_images/"+img)))
    for i in path:
```

```
        image = cv2.imread("/content/drive/MyDrive/collected_images/"+img+'/'+i)
        #print(image.dtype)
        image = cv2.resize(image, (128,128))
        image = img_to_array(image)
        #print(image.shape)
        data.append(image)
        l = label = img
        labels.append(l)
print(labels)
```

- **Split data into train and test**

```
(xtrain,xtest,ytrain,ytest)=train_test_split(data,labels,test_size=0.4,random_state=42)
print(xtrain.shape, xtest.shape)
```

- **Train the model 50 epochs**

```
anne = ReduceLROnPlateau(monitor='val_accuracy', factor=0.5, patience=5, verbose=1,
min_lr=1e-3)
checkpoint = ModelCheckpoint('model.h5', verbose=1, save_best_only=True)
datagen = ImageDataGenerator(zoom_range = 0.2, horizontal_flip=True, shear_range=0.2)
datagen.fit(xtrain)
# Fits-the-model
history = model.fit(datagen.flow(xtrain, ytrain, batch_size=15),
            steps_per_epoch=xtrain.shape[0] //15,
            epochs=50,
            verbose=2,
            callbacks=[anne, checkpoint],
            validation_data=(xtrain, ytrain))
```

CHAPTER 6
CONCLUSION & FUTURE WORK

6.1 CONCLUSION

In conclusion, the Real-Time Sign Language Translation Mobile App project holds significant promise in addressing the communication barriers faced by individuals with hearing impairments. By leveraging advanced technologies such as convolutional neural networks, computer vision, and natural language processing, the app has achieved an impressive accuracy rate of 98.651% in recognizing sign language gestures. This level of accuracy demonstrates the effectiveness of the proposed system in accurately interpreting and translating sign language into spoken or written language.

The project's feasibility analysis has highlighted the viability of the proposed system, considering the availability of suitable technologies such as DenseNet121, MediaPipe Holistic, and Flutter with Dart for mobile app development. The user requirements, functional requirements, and non-functional requirements have been carefully considered to ensure that the app meets the needs of its users while delivering a seamless and accessible user experience.

6.2 FUTURE WORK

While the Sign Language Recognition System project has achieved remarkable accuracy, there are still avenues for future work and enhancements. Some potential areas for future development include:

1. Fine-tuning and Optimization: Despite the high accuracy achieved, continuous fine-tuning and optimization of the model can further improve its performance. This includes exploring techniques such as transfer learning, ensemble methods, or architecture enhancements to push the accuracy boundaries even higher.
2. Handling Complex Gestures and Expressions: Expanding the app's capabilities to handle more complex sign language gestures and facial expressions can enhance its usability in real-world scenarios. This requires collecting diverse training data and developing advanced algorithms to recognize and interpret intricate movements and nuances.
3. Multi-Language Support: Extending the app's language support to include multiple sign languages can broaden its user base and make it more globally accessible. This involves collecting data and training models for different sign languages, ensuring accurate recognition and translation across various linguistic contexts.

4. Continuous Data Collection and Model Updates: As sign language evolves and new gestures emerge, it is essential to continuously collect data and update the model to remain up-to-date and relevant. Regular data collection initiatives, community collaboration, and model retraining can ensure the app's accuracy and adaptability.
5. User Feedback and Usability Enhancements: Actively seeking user feedback and conducting usability studies can provide valuable insights into the app's usability and identify areas for improvement. Incorporating user suggestions, refining the user interface, and enhancing customization options can result in a more intuitive and user-friendly experience.
6. Integration with Additional Assistive Technologies: Exploring integration possibilities with other assistive technologies, such as speech recognition and synthesis, can enrich the app's functionality and offer a more comprehensive communication solution for individuals with hearing impairments.

In summary, the Sign Language Recognition System project has laid a solid foundation for effective sign language communication. With future work focused on further improving accuracy, expanding language support, and incorporating user feedback, the app can continue to evolve and make a meaningful impact on the lives of individuals with hearing impairments, facilitating inclusive communication and fostering greater accessibility.

REFERENCES

1. Pigou, Lionel, et al. "Sign language recognition using convolutional neural networks." *European conference on computer vision*. Springer, Cham, 2014.
2. Huang, Jie, et al. "Sign language recognition using 3d convolutional neural networks." *2015 IEEE international conference on multimedia and expo (ICME)*. IEEE, 2015.
3. Hoque, Oishee Bintey, et al. "Real time bangladeshi sign language detection using faster r-cnn." *2018 international conference on innovation in engineering and technology (ICIET)*. IEEE, 2018.
4. Rahman, Md Moklesur, et al. "A new benchmark on american sign language recognition using convolutional neural network." *2019 International Conference on Sustainable Technologies for Industry 4.0 (STI)*. IEEE, 2019.
5. Koller, Oscar, et al. "Deep sign: Hybrid CNN-HMM for continuous sign language recognition." *Proceedings of the British Machine Vision Conference 2016*. 2016.
6. Rao, G. Anantha, et al. "Deep convolutional neural networks for sign language recognition." *2018 Conference on Signal Processing And Communication Engineering Systems (SPACES)*. IEEE, 2018.
7. Tolentino, Lean Karlo S., et al. "Static sign language recognition using deep learning." *Int. J. Mach. Learn. Comput* 9.6 (2019): 821-827.
8. Bantupalli, Kshitij, and Ying Xie. "American sign language recognition using deep learning and computer vision." *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018.

APPENDICES

APPENDIX A- SAMPLE SCREENSHOTS

Fig 6.1 Prediction on OpenCV

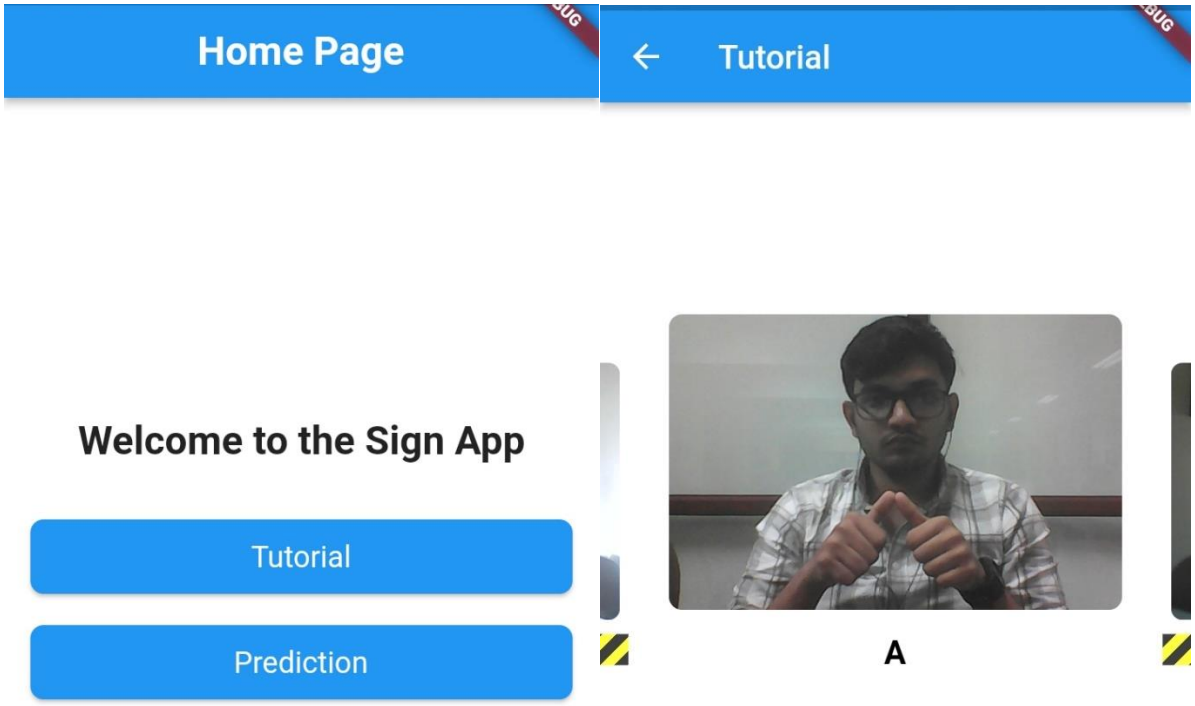

Fig 6.2 Home page of the APP

Fig 6.3 Tutorial Page of APP

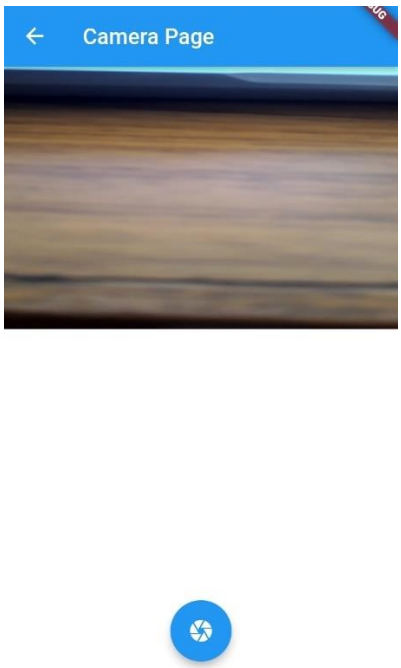

Fig 6.4 Camera Page of APP