An Industry Oriented Mini Project Report

on

Real-Time Sign Language Recognition For Enhanced Communication

Submitted in partial fulfillment of the requirements for the award of a degree of

Bachelor Of Technology

in

Computer Science And Engineering

By

Ch.Sweekruthi Reshma

(21EG505810)

L.Sreepadh Varma

(21EG505845)

T.Meghanath

(21EG505865)

Under The Guidance of

Mrs.B.Ujwala,

Asst. Professor, Department of CSE

Department of Computer Science and Engineering

ANURAG UNIVERSITY

Venkatapur (V), Ghatkesar (M), Medchal (D)., T.S-500088

(2023-2024)

DECLARATION

We hereby declare that the project work entitled **"Real-Time Sign Language Recognition For Enhanced Communication"** submitted to **Anurag University** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology (B. Tech)** in Computer Science and Engineering is a record of an original work done by us under the guidance of **Mrs. B. Ujwala, Assistant Professor** and this project work have not been submitted to any other university for the award of any other degree or diploma.

Ch.Sweekruthi Reshma

21EG505810

L.Sreepadh Varma

21EG505845

T.Meghanath

21EG505865

Date:

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the report entitled **"Real-Time Sign Language Recognition For Enhanced Communication"** being submitted by **Ch.Sweekruthi Reshma** bearing the Hall Ticket number **21EG505810**, **L.Sreepadh Varma** bearing the Hall Ticket number **21EG505845**, **T.Meghanath** bearing the Hall Ticket number **21EG505865** in partial fulfillment of the requirements for the award of the degree of the Bachelor of Technology in Computer Science and Engineering to Anurag University is a record of bonafide work carried out by them under my guidance and supervision.

The results embodied in this report have not been submitted to any other University for the award of any other degree.

Mrs. B. Ujwala
Assistant Professor

Dean, CSE

External Examiner

ACKNOWLEDGEMENT

We would like to express our sincere thanks and deep sense of gratitude to project supervisor **Mrs. B. Ujwala,**Assistant Professor, Dept of CSE for her constant encouragement and inspiring guidance without which this project could not have been completed. Her critical reviews and constructive comments improved our grasp of the subject and steered to the fruitful completion of the work. Her patience, guidance and encouragement made this project possible.

We would like to acknowledge our sincere gratitude for the support extended by **Dr. G. Vishnu Murthy**, Dean, Dept. of CSE, Anurag University. We also express our deep sense of gratitude to **Dr. V V S S S Balaram**, Academic Coordinator, **Dr. Pallam Ravi**, Project Coordinator and Project Review Committee members, whose research expertise and commitment to the highest standards continuously motivated us during the crucial stage of our project work.

We would like to express our special thanks to **Dr. V. Vijaya Kumar**, Dean School of Engineering, Anurag University, for his encouragement and timely support in our B.Tech program.

Ch.Sweekruthi Reshma **21EG505810**

L.Sreepadh Varma **21EG505845**

T.Meghanath **21EG505865**

ABSTRACT

Sign language is a crucial form of communication for the hearing-impaired community, allowing them to interact effectively with the world. Understanding sign language is challenging because it requires memorizing hand poses and gestures. This suggests a demand for an automatic sign language recognition system that allows everyone to understand this language. This mini project aims to develop a real-time sign language detection system that leverages computer vision, image classification and machine learning techniques to bridge the communication gap between the deaf and hearing communities. Including deep learning techniques, particularly convolutional neural networks (CNNs), the system interprets the sign language gestures captured by a camera in real-time. Using OpenCV libraries, the captured image undergoes preprocessing for feature extraction, enhancing the accuracy and robustness of the model. By integrating supervised learning, huge data collection, and more training, the proposed system results in accurately recognizing a large variety of sign language gestures in real-time scenarios. As a result, this model possesses potential to eliminate communication disparities, facilitating all-encompassing environment that promotes seamless interaction.

Keywords: Computer Vision, Image Classification, Convolution neural networks (CNN), Deep Learning Techniques, OpenCV, Supervised Learning.

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

In the ever-evolving landscape of technology, fostering inclusive communication is essential. Sign language serves as a vital medium for communication among the Deaf and Hard of Hearing (DHH) community. However, there exists a significant communication barrier between individuals who use sign language and those who do not understand it. To bridge this gap, advanced technologies like Computer Vision and Deep Learning have paved the way for real-time sign language recognition systems. Among these, Convolutional Neural Networks (CNN) have emerged as powerful tools, revolutionizing the way sign language is interpreted and enabling seamless interaction between diverse linguistic communities.

The Significance of Sign Language Recognition

Real-time sign language recognition has the potential to transform lives by enhancing communication for the DHH community. By accurately translating sign language gestures into text or speech, these systems break down barriers, enabling DHH individuals to communicate effectively with the broader society. This not only facilitates smoother daily interactions but also opens doors to education, employment, and social inclusion.

The Role of Convolutional Neural Networks (CNN)

CNNs have proven instrumental in sign language recognition due to their ability to automatically learn intricate patterns from visual data. These networks excel at recognizing complex gestures, capturing subtle nuances of hand movements, facial expressions, and body language inherent to sign language. By leveraging CNNs, real-time sign language recognition systems can achieve remarkable accuracy, making communication more intuitive and accessible.

Challenges and Opportunities

While significant progress has been made, challenges such as diverse signing styles, lighting conditions, and background clutter still persist. Addressing these challenges requires ongoing research and innovation. Moreover, the integration of real-time sign language recognition into everyday devices like smartphones and tablets holds immense promise. This integration could empower DHH individuals to communicate effortlessly, promoting inclusivity and understanding on a global scale.

In this context, this study explores the implementation of CNN-based real-time sign language recognition systems. By delving into the intricacies of CNN architectures, gesture recognition techniques, and innovative applications, this research aims to contribute to the advancement of technology-driven inclusive communication. Through this exploration, we endeavor to make significant strides in breaking down communication barriers, fostering understanding, and promoting a more inclusive society for all.

1.1 Motivation

Real-time sign language recognition holds immense significance and potential in addressing various societal needs and fostering inclusivity for the Deaf and Hard of Hearing (DHH) community. This technology goes far beyond merely translating sign language into text or speech; it has the potential to transform lives and bridge communication gaps.

Firstly, real-time sign language recognition can greatly enhance the educational experience for DHH individuals. By providing instant and accurate interpretation of sign language into written or spoken language, it can facilitate better access to mainstream educational resources. This empowers DHH students to actively participate in regular classrooms, ensuring they receive a quality education and reducing the need for specialized, segregated programs.

Furthermore, this technology has significant implications for the employment prospects of DHH individuals. Many job opportunities are restricted by communication barriers, but real-time sign language recognition can break down these walls. DHH individuals can more effectively communicate with colleagues and clients, potentially expanding their career opportunities and reducing unemployment rates within this community.

In addition to education and employment, real-time sign language recognition can also enhance access to healthcare. Many DHH individuals face challenges when trying to communicate their health concerns to medical professionals. Real-time recognition can ensure accurate communication, reducing the risk of misdiagnosis and improving overall healthcare outcomes.

Lastly, real-time sign language recognition has the potential to transform the way society perceives and interacts with the DHH community. It promotes inclusivity, fostering a sense of

belonging, and reducing stigmatization. As more individuals gain access to sign language communication, society can become more diverse, equal, and empathetic.

1.2 Problem Definition

The problem of real-time sign language communication poses a significant challenge due to the need for accurate and swift translation of complex hand gestures into understandable language. Leveraging Convolutional Neural Networks (CNNs) presents a solution to this challenge by enabling real-time interpretation of sign language gestures through advanced image recognition techniques. This technology aims to enhance communication for the deaf and hard of hearing, facilitating seamless interaction in various settings. By addressing the nuances of sign language, such as gestures' speed and intricacy, this approach strives to bridge the communication gap, promoting inclusivity, understanding, and accessibility for individuals with hearing impairments.

1.3 Objective

The objective of the project on real-time sign language for enhanced communication using CNN is to bridge the communication gap between hearing-impaired individuals and the rest of the world. The project aims to develop a robust and real-time system capable of recognizing and interpreting sign language gestures accurately. This technology strives to empower the deaf and hard of hearing community by providing them with a tool for seamless interaction in various settings, such as educational institutions and public spaces. The ultimate goal is to enhance inclusivity and facilitate effortless communication, fostering a more accessible and understanding society for all.

2. LITERATURE SURVEY

[1] J.Ekbote and M. Joshi endeavour to create an automated identification system for Indian Sign Language numerals ranging from zero to nine through their research work. The implementation database, which they created themselves, comprises 1000 images with 100 images per numeral sign. To extract the desired features, shape descriptors along with Scale Invariant Feature Transform (SIFT) and Histogram of Oriented Gradients (HOG) techniques are employed. The signs are classified using Artificial Neural Networks (ANN) and Support Vector Machine (SVM) classifiers, resulting in a remarkable accuracy rate of up to 99%.

[2] M. Jaiswal, V.Sharma, A.Sharma, S. Saini and R. Tomar have introduced a novel architecture for recognizing Indian Sign Language gestures made with two hands using binary values for weights and activations via bitwise operations to overcome difficulties associated with the recognition of sign language on specific embedded platforms. This binarized neural network has achieved an accuracy rate of 98.8%, surpassing existing methods; however, it struggles to correctly classify signs depicting M, N, and E due to their similar shapes while being limited in its ability to recognize only a small number of gesture classes as well.

[3] Raghuveera, T., Deepthi, R., Mangalashri, R. and their team, they were able to successfully identify Overlapping signs, double hand signs, and unique ISL (Indian Sign Language) signs through a new method. This project has shown an impressive improvement in average recognition accuracy of up to 71.85%. However, it is important to note that while the system achieved a perfect 100% accuracy for certain signs, it still struggled with accurately interpreting the context of gestures which led to erroneous translations on multiple occasions. Despite this limitation, the findings from this study represent a significant step forward in improving sign language recognition technology and could have important implications for the deaf and hard-of-hearing community.

[4] P. K. Athira, C. J. Sruthi, and A. Lijiya have developed a system that accurately identifies ISL gestures from mobile camera videos without the need for any additional sensors to detect hand regions. This innovative system can identify both single-handed static and dynamic gestures as well as double-handed static gestures with great precision using an ROI algorithm for feature extraction.The image is pre-processed before estimating the centroid of the binary image which

allows the system to recognize both static and dynamic gestures through its gesture separation algorithm.One major advantage of this economical system is that it can be easily implemented with a mobile camera, making it highly user-friendly but unfortunately not efficient under cluttered backgrounds or different illumination conditions.

[5] Rastgoo, Kiani, and Escalera presented an innovative system called "Video-based isolated hand sign language recognition using a deep cascaded model" that utilizes a cascaded architecture of SSD, CNN, and LSTM from RGB Videos to propose a sophisticated model for systematic hand sign recognition. This approach significantly enhances the accuracy and complexity of hand sign detection while enabling fast processing in uncontrolled environments with rapid hand motions. Further data can be incorporated to improve detection accuracy even further.

3.ANALYSIS

3.1 Existing System

The existing system for real-time sign language communication utilizes Convolutional Neural Networks to enhance communication between individuals using sign language. CNN technology enables efficient and accurate recognition of sign language gestures in real-time, bridging the gap between the deaf and hearing communities. By analyzing video input, the system interprets intricate hand movements and facial expressions, translating them into text or speech, facilitating seamless communication for the hearing-impaired. This innovative approach leverages advanced machine learning algorithms, allowing for natural, instant, and inclusive interactions, thereby promoting accessibility and understanding in diverse social contexts.

3.2 Proposed System

The proposed system aims to revolutionize communication for individuals with hearing impairments by leveraging advanced technology. It focuses on real-time sign language recognition using Convolutional Neural Networks, a deep learning technique specifically designed for image analysis. By employing CNNs, the system can accurately interpret and translate sign language gestures into text or speech, enabling seamless communication between deaf or hard-of-hearing individuals and the hearing world.

In real-time scenarios, the system captures sign language gestures through a camera or sensor. These gestures are then processed by the CNN model, which has been trained extensively on diverse sign language gestures to ensure robustness and accuracy. The CNN's ability to recognize intricate hand movements, facial expressions, and body language allows for precise interpretation.

The system's impact is profound, enhancing accessibility and inclusivity for the hearing-impaired community. It facilitates instant communication in various settings, such as education, healthcare, and social interactions. By bridging the communication gap, this innovative solution promotes equal opportunities and understanding for individuals with hearing impairments, fostering a more inclusive society. Through the integration of cutting-edge technology and deep learning, the proposed system empowers individuals with hearing impairments, enabling them to participate actively and meaningfully in the world around them.

3.3 System Requirement Specification

Operating system: Windows

IDE: Google Colab**,** VS Code

Language: Python

High processing machine for large data set, and libraries such as tensorflow, keras, cv2, numpy, matplotlib are utilized.

3.3.1 Purpose

The proposed system aims to enhance communication for the deaf and hard of hearing by providing seamless, accurate, and instant translation of sign language gestures into text or speech. This technology facilitates real-time communication between sign language users and non-signers, breaking down communication barriers in various settings such as education, healthcare, and social interactions. By enabling immediate and precise interpretation of sign language gestures, CNN-based systems empower individuals with hearing disabilities to engage effectively with the broader society, fostering inclusivity, understanding, and equal access to information and opportunities.

3.3.2 Scope

The scope of real-time sign language recognition through Convolutional Neural Networks (CNN) is profound, revolutionizing communication for the hearing-impaired. CNN's ability to process visual data makes it pivotal in capturing intricate hand gestures and facial expressions, crucial in sign language. Enhanced accuracy and speed in recognizing signs enable seamless real-time translation, fostering inclusivity. This technology finds applications in diverse fields such as education, customer service, and accessibility tools. Moreover, continuous advancements in CNN algorithms and hardware accelerate its implementation, promising a future where communication barriers are dismantled, promoting a more inclusive society for the hearing-impaired.

3.3.3 Overall Description

The Real-time sign language recognition leveraging Convolutional Neural Networks (CNNs) represents a transformative advancement in communication accessibility. This innovative system utilizes CNNs to process live sign language gestures captured by cameras, converting intricate hand movements and expressions into real-time text or speech. By harnessing deep learning algorithms, this technology interprets signs accurately, enabling seamless interaction between individuals with hearing impairments and those without. It enhances inclusivity, breaking

communication barriers, and fostering a more inclusive society. Real-time sign language recognition using CNNs revolutionizes how people connect, ensuring equal opportunities and empowering the deaf and hard of hearing communities in various social and professional contexts.

4.DESIGN

Designing a real-time sign language recognition system for enhanced communication using CNN involves employing deep learning techniques to interpret gestures accurately and instantaneously. CNNs are particularly suited for this task due to their ability to extract meaningful features from visual data. Here is a concise overview of the design process:

1. Data Collection and Preprocessing:
Gather a diverse dataset of sign language gestures, capturing various hand shapes, movements, and facial expressions. Preprocess the data by normalizing images, augmenting the dataset to enhance its size and variety, and labeling gestures for supervised learning.

2. CNN Architecture:
Design a CNN architecture tailored for sign language recognition. A typical architecture includes convolutional layers for feature extraction, pooling layers for spatial down-sampling, and fully connected layers for classification. Experiment with different architectures and hyperparameters to optimize accuracy and speed.

3. Real-time Processing:
Optimize the model for real-time performance by reducing computational complexity. Techniques like model quantization, pruning, and optimizing layers can speed up inference without compromising accuracy.

4. Gesture Recognition:
Implement real-time gesture recognition using the trained CNN model. Utilize techniques like frame differencing and motion detection to identify and track hand movements in video streams. Process these frames through the CNN to recognize the gestures accurately.

5. Integration and User Interface:
Integrate the real-time sign language recognition module into a user-friendly interface. This interface can be a mobile app, a web application, or a wearable device. Provide feedback to users in the form of text or synthesized speech, enabling seamless communication between signers and non-signers.

6. Continuous Learning and Improvement:

Implement mechanisms for continuous learning, allowing the system to improve its accuracy over time with additional data. Employ techniques like transfer learning and online learning to adapt the model to new gestures and variations in signing styles.

By combining these steps and leveraging the power of CNNs, a real-time sign language recognition system can enhance communication for individuals with hearing impairments, promoting inclusivity and enabling a more accessible world.

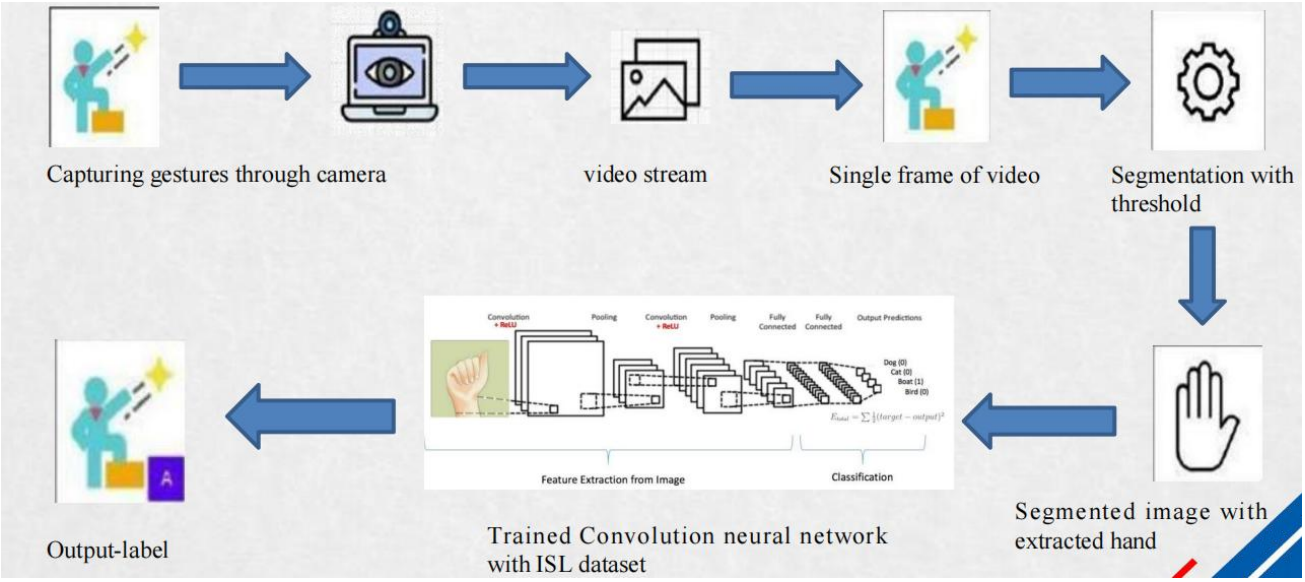


Fig 4 System Architecture

The system architecture shown in the image is a generic one for capturing gestures through device cameras. It consists of the following components:

- **Device camera:** This is the camera that is used to capture the gestures. It can be a built-in camera on a device, such as a smartphone or laptop, or an external camera.
- **Video stream:** The video stream from the camera is sent to the computer for processing.
- **Hand segmentation:** The first step in processing the video stream is to segment the hand from the background. This can be done using a variety of computer vision techniques, such as thresholding, skin color detection, and edge detection.

- **Feature extraction:** Once the hand has been segmented, a variety of features can be extracted from the image, such as the position of the fingers, the orientation of the palm, and the shape of the hand.

- **Gesture recognition:** The extracted features are then used to classify the gesture. This can be done using a variety of machine learning techniques, such as support vector machines (SVMs), random forests, and convolutional neural networks (CNNs).

- **Output:** The output of the gesture recognition system can be a variety of things, such as a text label, a control signal, or a 3D model of the hand.

The specific implementation of each component of the system will vary depending on the specific application. For example, the hand segmentation algorithm may need to be different for different types of cameras and different lighting conditions. Similarly, the gesture recognition algorithm may need to be different for different types of gestures.

4.1. UML Diagrams

UML stands for Unified Modeling Language. Taking SRS document of analysis as input to the design phase drawn UML diagrams. The UML is only language so is just one part of the software development method. The UML is process independent, although optimally it should be used in a process that should be driven, architecture-centric, iterative, and incremental. The UML is language for visualizing, specifying, constructing, documenting the articles in a software- intensive system. It is based on diagrammatic representations of software components. A modeling language is a language whose vocabulary and rules focus on the conceptual and physical representation of the system. A modeling language such as the UML is thus a standard language for software blueprints. The UML is a graphical language, which consists of all interesting systems. There are also different structures that can transcend what can be represented in a programming language.

4.1.1 USE CASE Diagram:

Use Case during requirement elicitation and analysis to represent the functionality of the system. Use case describes a function by the system that yields a visible result for an actor.

Definitions of the boundary of the system i.e., differentiating the tasks accomplished by the system and the tasks accomplished by its environment. The actors are outside the boundary of the system, whereas the use cases are inside the boundary of the system. Use case describes the behavior of the system as seen from the actor's point of view. It describes the function provided by the system as a set of events that yield a visible result for the actor.

Purpose of Use Case Diagrams:The purpose of use case diagram is to capture the dynamic aspect of a system. However, this definition is too generic to describe the purpose, as other four diagrams (activity, sequence, collaboration, and Statechart) also have the same purpose. We will look into some specific purpose, which will distinguish it from other diagrams. Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. Hence, when a system is analyzed to gather its functionalities, use cases are prepared and actors are identified. When the initial task is complete, use case diagrams are modelled to present the outside view. In brief, the purposes of use case diagrams can be said to be as follows −

- Used to gather the requirements of a system.
- Used to get an outside view of a system.

- Identify the external and internal factors influencing the system.
- Show the interaction among the requirements are actors.

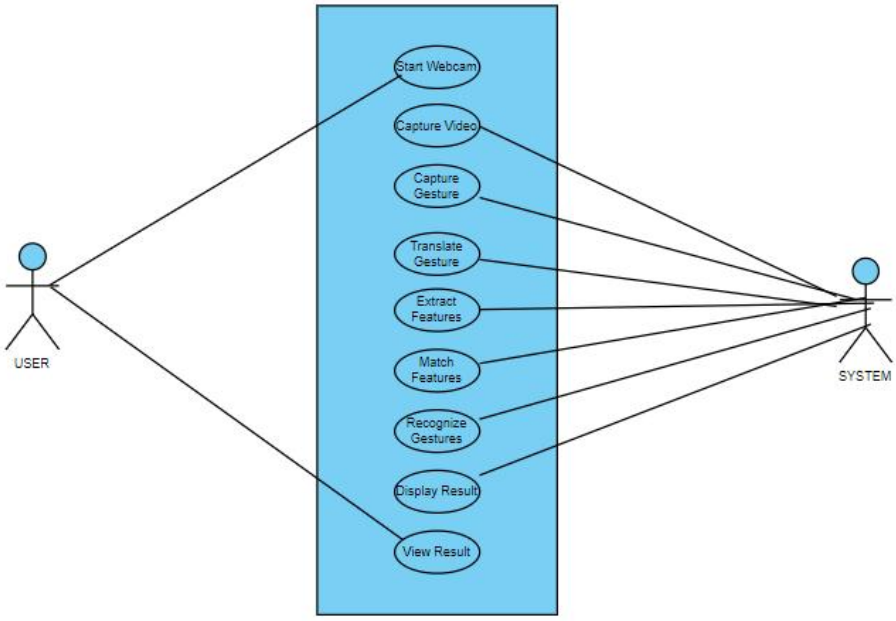


Fig 4.1.1. Use Case Diagram For Sign Language Recognition

4.2 Data Flow Diagrams

The DFD is also known as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of the input data to the system, various processing carried out on these data, and the output data is generated by the system. It maps out the flow of information for any process or system, how data is processed in terms of inputs and outputs. It uses defined symbols like rectangles, circles and arrows to show data inputs, outputs, storage points and the routes between each destination. They can be used to analyse an existing system or model of a new one. A DFD can often visually "say" things that would be hard to explain in words and they work for both technical and non- technical.

There are four components in DFD:

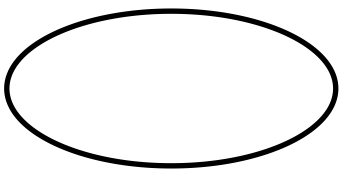
1. External Entity
2. Process
3. Data Flow
4. Data Store

1) External Entity: It is an outside system that sends or receives data, communicating with the system. They are the sources and destinations of information entering and leaving the system. They might be an outside organization or person, a computer system or a business system. They are known as terminators, sources and sinks or actors. They are typically drawn on the edges of the diagram. These are sources and destinations of the system's input and output.

Representation:

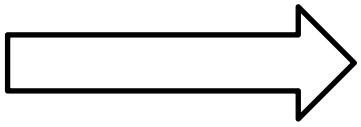
2) Process: It is just like a function that changes the data, producing an output. It might perform computations for sort data based on logic or direct the dataflowbased on business rules.

Representation:



3) Data Flow: A dataflow represents a package of information flowing between two objects in the data-flow diagram, Data flows are used to model the flow of information into the system, out of the system and between the elements within the system.

Representation:



4) Data Store: These are the files or repositories that hold information for later use, such as a database table or a membership form. Each data store receives a simple label.

Representation:

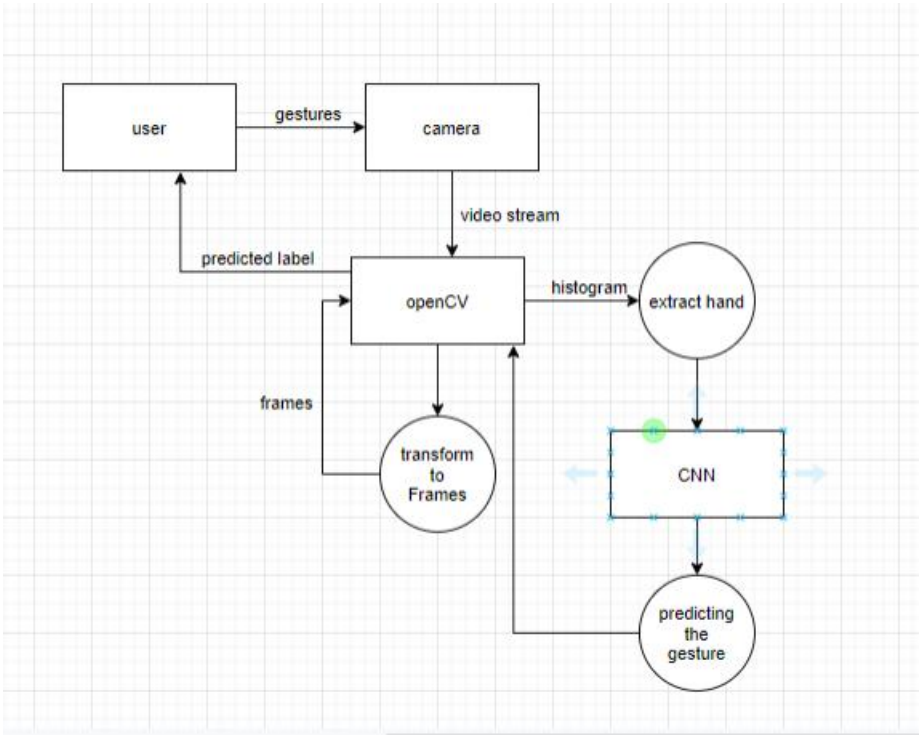


Fig 4.2 Data Flow Diagram

4.3 Sequence Diagram:

Sequence diagram displays the time sequence of the objects participating in the interaction. This consists of the vertical dimension(time) and horizontal dimension (different objects). Objects: Object can be viewed as an entity at a particular point in time with specific value and as a holder of identity.

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

A sequence diagram shows, as parallel vertical lines (lifelines), different processesor objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

If the lifeline is that of an object, it demonstrates a role. Leaving the instance name blank can represent anonymous and unnamed instances. Messages, written with horizontal arrows with the message name written above them, display interaction. Solid arrow heads represent synchronous calls, open arrow heads represent asynchronous messages, and dashed lines represent reply messages. If a caller sends a synchronous message, it must wait until the message is done, such as invoking a subroutine. If a caller sends an asynchronous message, it can continue processing and doesn’t have to wait for a response. Asynchronous calls are present in multithreaded applications, event-driven applications and in message-oriented middleware. Activation boxes, or method-call boxes, are opaque rectangles drawn on top of lifelines to represent that processes are being performed in response to the message (ExecutionSpecifications in UML).

Objects calling methods on themselves use messages and add new activation boxes on top of any others to indicate a further level of processing. If an object is destroyed (removed from memory), an X is drawn on bottom of the lifeline, and the dashed line ceases to be drawn below it. It should be the result of a message, either from the object itself, or another.

A message sent from outside the diagram can be represented by a message originating from a filled-in circle (found message in UML) or from a border of the sequence diagram (gate in UML). UML has introduced significant improvements to the capabilities of sequence diagrams. Most of these improvements are based on the idea of interaction fragments which represent smaller pieces of an enclosing interaction. Multiple interaction fragments are combined to create a variety of combined fragments, which are then used to model interactions that include parallelism, conditional branches, optional interactions

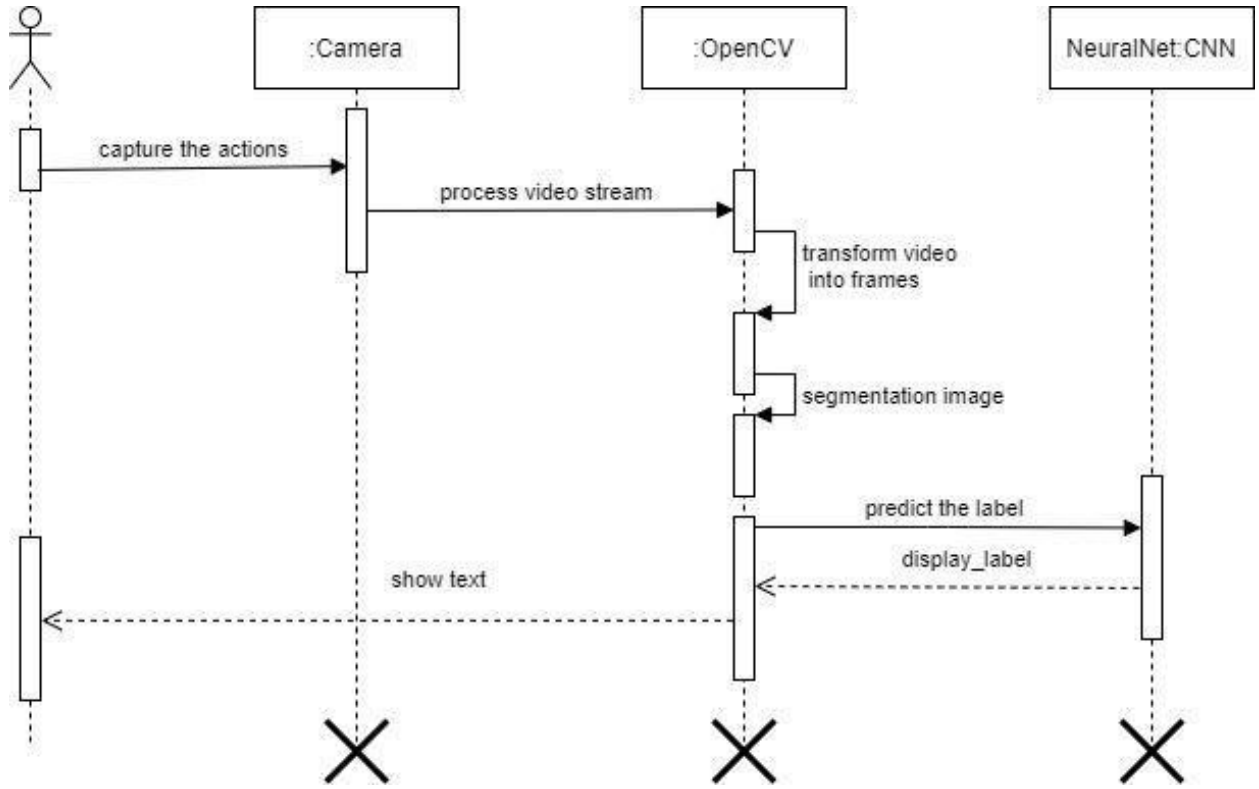


Fig 4.3 Sequence Diagram of Sign Language Recognition

4.4 State Chart

A state chart diagram describes a state machine which shows the behaviour of classes. It shows the actual changes in state not processes or commands that create those changes and is the dynamic behaviour of objects over time by model1ing the life cycle of objects of each class. It describes how an object is changing from one state to another state. There are mainly two states in State Chart Diagram:1. Initial State 2. Final-State. Some of the components of State Chart Diagram are:

State: It is a condition or situation in life cycle of an object during which it's satisfies same condition or performs some activity or waits for some event.

Transition: It is a relationship between two states indicating that object in first state performs some actions and enters into the next state or event.

Event: An event is specification of significant occurrence that has a location in time and space.

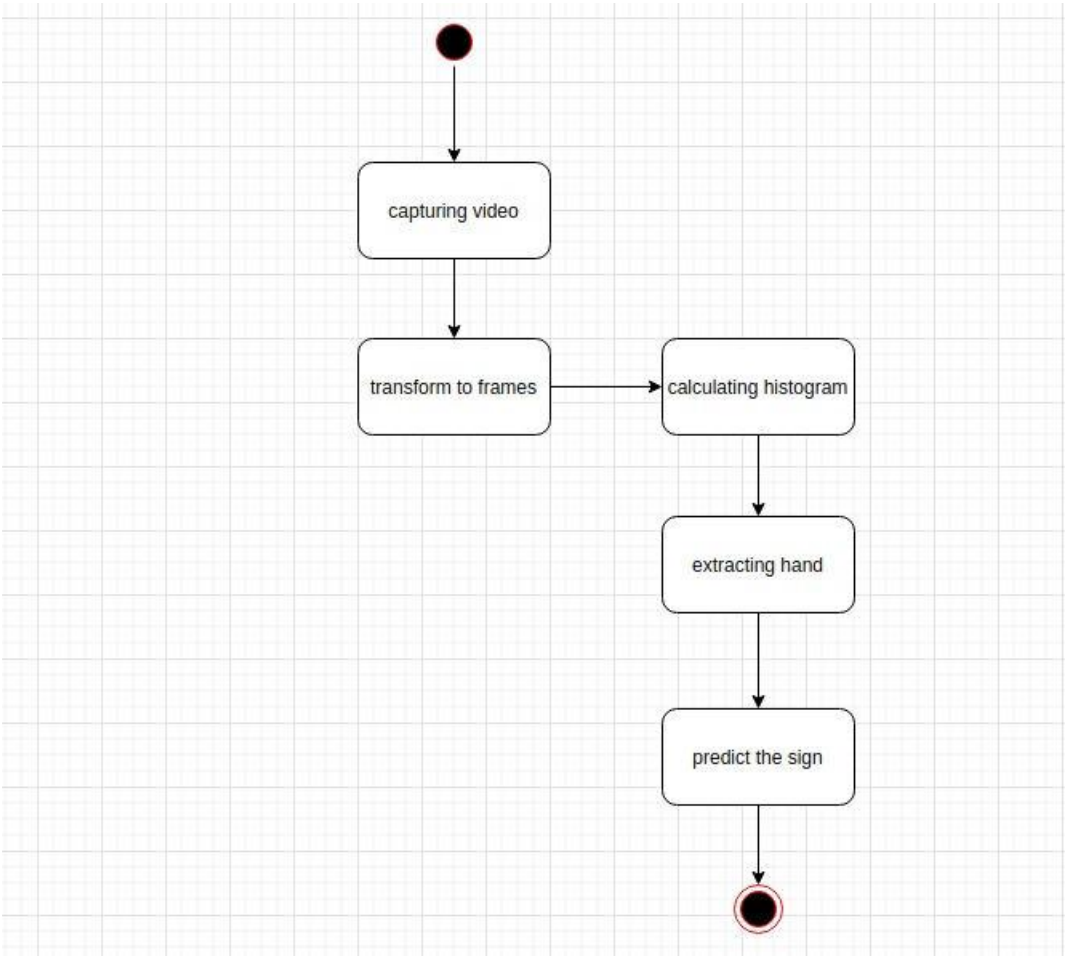


Fig 4.4 State Chart For Sign Language Recognition

5.IMPLEMENTATION

Real-time sign language recognition is a fascinating application of computer vision and machine learning that aims to bridge the communication gap between the deaf and hearing communities. Convolutional Neural Networks (CNNs) have proven to be highly effective in this context, as they can be tailored to extract features from visual sign language gestures and classify them in real-time. The implementation of real-time sign language recognition using CNNs involves several crucial steps, which we will outline in the following paragraphs.

The first step in building a real-time sign language recognition system is data collection and preprocessing. High-quality sign language datasets containing a variety of signs and gestures are essential. Each sign gesture must be recorded and labeled, with appropriate hand tracking techniques to capture the dynamic nature of sign language. Data augmentation techniques can be applied to enhance the dataset's diversity and improve model generalization.

Once the dataset is ready, the next step is to design the CNN architecture. A convolutional neural network is a deep learning model that excels at feature extraction from images. The architecture typically comprises multiple convolutional layers, pooling layers, and fully connected layers. Researchers often experiment with different CNN architectures, such as LeNet, VGG, or more complex models like ResNet and Inception, to find the best-suited architecture for sign language recognition. The choice of the architecture depends on the complexity of the sign language dataset and the desired real-time performance.

Training the CNN model is the next crucial phase. This involves splitting the dataset into training and validation sets and feeding it into the CNN. The network learns to recognize patterns and features within the sign language gestures during this training process. Various optimization techniques and loss functions are employed to fine-tune the model's weights and biases, ensuring accurate sign recognition. Real-time processing requirements demand optimization for inference speed, often requiring the use of hardware acceleration or model quantization.

5.1 Module

Supervised machine learning: It is one of the ways of machine learning where the model is trained by input data and expected output data. To create such model, it is
necessary to go through the following phases:

1. model construction
2. model training
3. model testing
4. model evaluation

5.2 Module Description

1.Model Construction:

The implementation of deep learning algorithms, particularly neural networks, plays a crucial role in various projects, including the development of real-time sign language recognition systems.

1. Model Initialization: The process begins by initializing a neural network model. In this case, Keras, a popular high-level neural network API, is used to define the model. This is often done as follows: `model = Sequential()`. This sets up a linear stack of layers for building the neural network.

2. Layer Definition: Neural networks consist of multiple layers, and it's essential to specify the types of layers and their configurations. Layers are added to the model using the `model.add(type_of_layer())` method. These layers can include input layers, convolutional layers, pooling layers, fully connected layers, and more. The choice of layers and their arrangement depends on the specific requirements of the problem.

3. Model Compilation: Once the layers are defined, the model needs to be compiled. During this compilation step, Keras interfaces with TensorFlow (a popular machine learning library) to build the computational graph for the model. Important parameters to specify during compilation include the loss function and the optimizer algorithm. The loss function, specified as `loss='name_of_loss_function'`, measures how well the model's predictions align with the actual target values. The optimizer, specified as `optimizer='name_of_optimizer_alg'`, is responsible for adjusting the model's internal parameters to minimize the loss function.

2. Model Training: In the realm of deep learning, the process of model training is considered to be a crucial stage. During this phase, the neural network that has been constructed is put through its paces using training data and their corresponding expected outputs. This is achieved by utilizing the `model.fit(training_data, expected_output)` function, which allows for the tracking of training progress in real-time via console output. Upon completion of training, the model then reports its final accuracy score, which serves as a valuable metric for measuring how well it has learned to make predictions based on the provided training data. All in all, model training plays an integral role in ensuring that deep learning algorithms are able to function effectively and accurately in a variety of different contexts and applications. After confirming that the model is giving good accuracy, it can be saved for future use with the command `model.save("name_of_file.h5")`. This saved model can then be deployed and utilized in real-world applications, enabling it to evaluate and make predictions on new, unseen data, marking the transition from model development to practical use.

3. Model Testing :In real-time sign language recognition, model testing involves evaluating the trained neural network's performance on a separate dataset of sign language gestures captured in real-time. As video frames are continuously fed into the model for inference, recognition results are compared with ground truth labels to calculate performance metrics such as accuracy, precision, recall, and F1-score. These results are typically displayed in real-time through a graphical user interface or overlaid on the video feed.

4. Model Evaluation: Model evaluation in the context of sign language recognition using neural networks is a critical phase aimed at assessing the model's performance, generalization capabilities, and readiness for real-world applications. This evaluation process involves several key steps and considerations.The first step in model evaluation is to acquire a separate dataset that was not used during the training phase. This dataset typically contains sign language gestures and movements captured from various sources or individuals. It is crucial that this evaluation dataset is diverse and representative of the sign language signs and gestures that the model is expected to recognize in real-world scenarios.

Once the evaluation dataset is prepared, it is used to assess the model's accuracy and effectiveness. The model is fed with the evaluation data, and its predictions are compared to the ground truth labels or annotations. Common performance metrics, such as accuracy, precision,

recall, and F1-score, are calculated to quantify the model's recognition capabilities. These metrics provide insights into the model's ability to correctly identify sign language gestures.

In addition to evaluating accuracy, the model's robustness and generalization are tested. This involves assessing how well the model performs on signs and gestures that it has not encountered before, measuring its ability to adapt to variations in sign execution, different signers, and environmental conditions. The model should demonstrate an ability to generalize and recognize a wide range of signs accurately.

User feedback and usability testing are integral components of model evaluation. Real-world usability and user experience are crucial for the success of sign language recognition systems. The interface through which users interact with the system is evaluated for its user-friendliness, responsiveness, and overall effectiveness in facilitating communication between sign language users and those who may not be familiar with sign language.

System latency, or the delay between the presentation of a sign and the model's recognition of that sign, is also assessed during model evaluation. Reducing latency is essential for real-time communication, ensuring that the system responds quickly to user gestures.

Furthermore, model evaluation is not a one-time process; it should be ongoing. As the system is deployed in real-world scenarios, continuous monitoring and feedback collection are necessary to identify and address any limitations, false positives, false negatives, and areas for improvement. This iterative approach to model evaluation ensures that the sign language recognition system evolves to meet the needs of its users and remains accurate and reliable in practical applications.

5.3 Introduction to Technologies used:

IDLE (short for Integrated Development and Learning Environment) is an integrated development environment for Python, which has been bundled with the default implementation of the language since 1.5.2b. It is packaged as an optional part of the Python packaging with many Linux distributions. It is completely written in Python and the Tkinter GUI toolkit (wrapper functions for Tcl/Tk).

IDLE is intended to be a simple IDE and suitable for beginners, especially in an educational environment. To that end, it is cross-platform, and avoids feature clutter.

Visual Studio is a popular integrated development environment (IDE) created by Microsoft for software development. It provides a robust and versatile set of tools for building a wide range of applications, including web, mobile, desktop, cloud, and game development. Visual Studio offers a rich, user-friendly interface and supports multiple programming languages, making it a preferred choice for developers across the globe.

One of the key features of Visual Studio is its code editor, which is highly customizable and supports features like syntax highlighting, code completion, and debugging. It also offers a vast collection of extensions through the Visual Studio Marketplace, allowing developers to enhance their development environment with various plugins and tools.

Visual Studio supports a wide array of programming languages, including C#, C++, Python, JavaScript, and more. It provides built-in project templates and tools for specific frameworks and platforms, such as .NET, Xamarin, and Azure, making it easier for developers to get started on their projects.

In addition to coding, Visual Studio includes powerful debugging and testing tools, enabling developers to identify and fix issues efficiently. It also supports version control systems like Git for collaboration and code management. The IDE seamlessly integrates with Azure DevOps for end-to-end application lifecycle management, including continuous integration and continuous delivery (CI/CD).

Visual Studio is available in several editions, with Visual Studio Community being the free version, and Visual Studio Professional and Visual Studio Enterprise offering more advanced features and capabilities. Whether you're a beginner or an experienced developer, Visual Studio provides a comprehensive development environment to help you create high-quality software and applications.

Deep learning neural networks are a subset of artificial neural networks inspired by the structure and function of the human brain. They have gained significant prominence in recent years due to their remarkable ability to solve complex tasks, particularly in the fields of computer vision, natural language processing, and reinforcement learning.

These networks consist of multiple layers of interconnected nodes or neurons, organized into an input layer, hidden layers, and an output layer. Each neuron performs a weighted sum of its inputs, applies an activation function, and passes the result to the next layer. This hierarchical structure allows deep learning models to learn and represent intricate patterns and features in data, making them well-suited for tasks that involve high-dimensional and unstructured data, such as images and text.

One of the key strengths of deep learning is its capacity to automatically discover relevant features from raw data, eliminating the need for manual feature engineering. This characteristic, combined with the availability of large datasets and powerful hardware, has led to significant advancements in various applications. Convolutional neural networks (CNNs) excel in image analysis, while recurrent neural networks (RNNs) are well-suited for sequential data processing. Deep learning models can also be combined into more complex architectures, such as recurrent-convolutional networks or transformers, to handle diverse tasks.

Training deep learning models typically involves using a loss function and an optimization algorithm to iteratively adjust the network's parameters (weights and biases) to minimize the error between predicted and actual outputs. This process, known as backpropagation, requires a large amount of labeled training data and substantial computational resources. Transfer learning is a common strategy in deep learning, where pre-trained models are fine-tuned on specific tasks to leverage existing knowledge.

Deep learning has found applications in numerous domains, including image and speech recognition, natural language understanding and generation, autonomous vehicles, recommendation systems, and healthcare diagnostics. Despite their success, deep learning models can be data-hungry and computationally expensive, and they often lack transparency and interpretability, which are ongoing research challenges. However, their potential for solving complex problems and their ability to continually improve with larger datasets and more efficient hardware make deep learning neural networks a crucial technology in the field of artificial intelligence.

Convolution Neural Network:

Image classification is the process of taking an input(like a picture) and out putting its class or probability that the input is a particular class. Neural networks are applied in the following steps:

1) One hot encode the data: A one-hot encoding can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer value.

2) Define the model: A model said in a very simplified form is nothing but a function that is used to take in certain input, perform certain operation to its beston the given input (learning and then predicting/classifying) and produce the suitable output.

3) Compile the model: The optimizer controls the learning rate. We will be using ‘adam’ as our optmizer. Adam is generally a good optimizer to use for many cases. The adam optimizer adjusts the learning rate throughout training. The learning rate determines how fast the optimal weights for the model are calculated. A smaller learning rate may lead to more accurate weights (up to a certain point), but the time it takes to compute the weights will be longer.

4) Train the model: Training a model simply means learning (determining) good values for all the weights and the bias from labeled examples. In supervised learning, a machine learning algorithm builds a model by examining many examples and attempting to find a model that minimizes loss; this process is called empirical risk minimization.

5) Test the model : A convolutional neural network convolves learned featured with input data and uses 2D convolution layers.

ConvolutionOperation: In purely mathematical terms, convolution is a function derived from two given functions by integration which expresses how the shape of one is modified by the other. Here are the three elements that enter into the convolution operation:

- Input image
- Feature detector
- Feature map Steps to apply convolution layer
- You place it over the input image beginning from the top-left corner within the borders you see demarcated above, and then you count the number of cells in which the feature detector matches the input image.
- The number of matching cells is then inserted in the top-left cell of the feature map
- You then move the feature detector one cell to the right and do the same thing. This movement is called a and since we are moving the feature detector one cell at time, that would be called a stride of one pixel.
- What you will find in this example is that the feature detector's middle-left cell with the number 1 inside it matches the cell that it is standing over inside the input image. That's the only matching cell, and so you write “1” in the next cell in the feature map, and so on and so forth.
- After you have gone through the whole first row, you can then move it over to the next row and go through the same process.

There are several uses that we gain from deriving a feature map. These are the most important of them: Reducing the size of the input image, and you should know that the larger your strides (the movements across pixels), the smaller your feature map.

Relu Layer: Rectified linear unit is used to scale the parameters to non negativevalues.We get pixel values as negative values too . Inthis layer we make them as 0’s. The purpose of applying the rectifier function is to increase the non-linearity in our images. The reason we want to do that is that images are naturally non-linear. The rectifier serves to break up the linearity even further in order to make up for the linearity that we might impose an image when we put it through the convolution operation. What the rectifier function does to an image like this is remove all the black elements from it, keeping only those carrying a positive value (the grey and white colors).The essential difference between the non-rectified version of the image and the rectified one is the progression of colors. After we rectify the image, you will find the colors changing more abruptly. The gradual change is no longer there. That indicates that the linearity has been disposed of.

Pooling Layer: The pooling (POOL) layer reduces the height and width of the input. It helps reduce computation, as well as helps make feature detectors more invariant to its position in the input This process is what provides the convolutional neural network with the “spatial variance” capability. In addition to that, pooling serves to minimize the size of the images as well as the number of parameters which, in turn, prevents an issue of “overfitting” from coming up. Overfitting in a nutshell is when you create an excessively complex model in order to account for the idiosyncracies we just mentioned The result ofusing a pooling layer and creating down sampled or pooled feature maps is a summarized version of the features detected in the input. They are useful as small changes in the location of the feature in the input detected by the convolutional layer will result in a pooled feature map with the feature in the same location. Thiscapability added by pooling is called the model’s invariance to local translation.

Fully Connected Layer: The role of the artificial neural network is to take this data and combine the features into a wider variety of attributes that make the convolutional network more capable of classifying images, which is the whole purpose from creating a convolutional neural network. It has neurons linked to each other ,and activates if it identifies patterns and sends signals to output layer .the outputlayer gives output class based on weight values, For now, all you need to know is that the loss function informs us of how accurate our network is, which we then use in optimizing our network in order to increase its effectiveness. That requires certain things to be altered in our network. These include the weights (the blue lines connecting the neurons, which are basically the synapses), and the feature detector since the network often turns out to be looking for the wrong features and has to be reviewed multiple times for the sake of optimization.This full connection process practically works as follows:

- The neuron in the fully-connected layer detects a certain feature; say, a nose.
- It preserves its value.
- It communicates this value to the classes trained images.

5.4 Sample Code

Dataset Collection Code For Single Hand Gestures:

```
import cv2
from cvzone.HandTrackingModule import HandDetector
import numpy as np
import math
import time
cap = cv2.VideoCapture(0)
detector = HandDetector(maxHands=2)
offset = 20
imgSize = 300
folder = "Data/L"
counter = 0
while True:
    success, img = cap.read()
    hands, img = detector.findHands(img)
    if hands:
        hand = hands[0]
        x, y, w, h = hand['bbox']
        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255
        imgCrop = img[y - offset:y + h + offset, x - offset:x + w + offset]
        imgCropShape = imgCrop.shape
        aspectRatio = h / w
        if aspectRatio > 1:
            k = imgSize / h
            wCal = math.ceil(k * w)
            imgResize = cv2.resize(imgCrop, (wCal, imgSize))
            imgResizeShape = imgResize.shape
            wGap = math.ceil((imgSize - wCal) / 2)
            imgWhite[:, wGap:wCal + wGap] = imgResize
        else:
            k = imgSize / w
            hCal = math.ceil(k * h)
```

```
            imgResize = cv2.resize(imgCrop, (imgSize, hCal))
            imgResizeShape = imgResize.shape
            hGap = math.ceil((imgSize - hCal) / 2)
            imgWhite[hGap:hCal + hGap, :] = imgResize
        cv2.imshow("ImageCrop", imgCrop)
        cv2.imshow("ImageWhite", imgWhite)
    cv2.imshow("Image", img)
    key = cv2.waitKey(1)
    if key == ord("s"):
        counter += 1
        cv2.imwrite(f'{folder}/Image_{time.time()}.jpg',imgWhite)
        print(counter)
```

Dataset Collection Code for Double-Hand Gestures:

```
import cv2
from cvzone.HandTrackingModule import HandDetector
import numpy as np
import math
import time
cap = cv2.VideoCapture(0)
detector = HandDetector(maxHands=2)  # Track up to two hands
offset = 20
imgSize = 300
folder = "Data/Z"
counter = 0
while True:
    success, img = cap.read()
    hands, img = detector.findHands(img)
    if hands:
        # Find a bounding box that encompasses both hands
        x_min, y_min, x_max, y_max = float('inf'), float('inf'), -float('inf'), -float('inf')
        for hand in hands:
            x, y, w, h = hand['bbox']
            x_min = min(x_min, x - offset)
            y_min = min(y_min, y - offset)
```

```
    x_max = max(x_max, x + w + offset)
    y_max = max(y_max, y + h + offset)

# Ensure that the bounding box coordinates are within the image boundaries
x_min = max(x_min, 0)
y_min = max(y_min, 0)
x_max = min(x_max, img.shape[1])
y_max = min(y_max, img.shape[0])

# Crop the region containing both hands
imgCrop = img[y_min:y_max, x_min:x_max]

imgCropShape = imgCrop.shape

aspectRatio = imgCropShape[1] / imgCropShape[0]

if aspectRatio > 1:
    k = imgSize / imgCropShape[1]
    wCal = math.ceil(k * imgCropShape[1])
    imgResize = cv2.resize(imgCrop, (wCal, imgSize))
    imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255
    wGap = math.ceil((imgSize - wCal) / 2)
    imgWhite[:, wGap:wCal + wGap] = imgResize

else:
    k = imgSize / imgCropShape[0]
    hCal = math.ceil(k * imgCropShape[0])
    imgResize = cv2.resize(imgCrop, (imgSize, hCal))
    imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255
    hGap = math.ceil((imgSize - hCal) / 2)
    imgWhite[hGap:hCal + hGap, :] = imgResize

cv2.imshow("ImageCrop", imgCrop)
cv2.imshow("ImageWhite", imgWhite)
```

```
    cv2.imshow("Image", img)
    key = cv2.waitKey(1)
    if key == ord("s"):
        counter += 1
        cv2.imwrite(f'{folder}/Image_{time.time()}.jpg', imgWhite)
        print(counter)
    if key == ord("q"):  # Press 'q' to exit the loop
        break
cap.release()
cv2.destroyAllWindows()
```

Training Code:

```
import numpy as np
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D,Dense,Flatten,Dropout
import matplotlib.pyplot as plt
from keras.layers import BatchNormalization
from keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from google.colab import drive
drive.mount('/content/gdrive')

train_dir="/content/gdrive/MyDrive/sign/data"
generator = ImageDataGenerator()
train_ds = generator.flow_from_directory(train_dir,target_size=(224, 224),batch_size=32)
classes = list(train_ds.class_indices.keys())

from tensorflow.keras.preprocessing.image import ImageDataGenerator

image_gen = ImageDataGenerator(
    rescale=1./255,
```

```
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2  # 20% of the data will be used for validation
)

train_data_gen = image_gen.flow_from_directory(
    batch_size=32,
    directory=train_dir,
    target_size=(224, 224),
    class_mode='categorical',
    subset='training'
)

val_data_gen = image_gen.flow_from_directory(
    batch_size=32,
    directory=train_dir,
    target_size=(224, 224),
    class_mode='categorical',
    subset='validation'
)
# This function will plot images in the form of a grid with 1 row and 5 columns where images are
placed in each column.
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 5, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
        ax.imshow(img)
    plt.tight_layout()
    plt.show()
```

```
# Retrieve a batch of augmented images
augmented_images = [train_data_gen[0][0][i] for i in range(5)]

# Plot the augmented images using the plotImages function
plotImages(augmented_images)
model = Sequential()
model.add(Conv2D(32, kernel_size = (3, 3), activation='relu', input_shape=(224,224,3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(96, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
#model.add(Dropout(0.3))
model.add(Dense(len(classes),activation='softmax'))
model.compile(loss = 'categorical_crossentropy',optimizer = 'adam',metrics = ["accuracy"])
model.summary()
epochs=5
history=model.fit(train_data_gen,steps_per_epoch=int(np.ceil(8672/float(32))),epochs=epochs,va
lidation_data=val_data_gen,validation_steps=int(np.ceil(2168/ float(32))))
```

```
final_train_accuracy = history.history['accuracy'][-1]
print(f'Final training accuracy: {final_train_accuracy * 100:.2f}%')

model.save('keras_model.h5')
```

Testing Code:

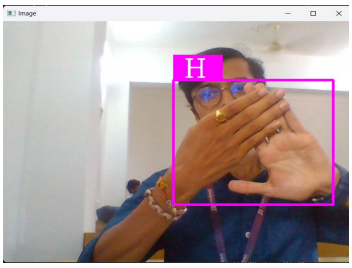
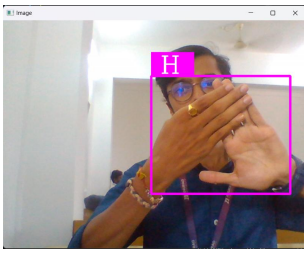
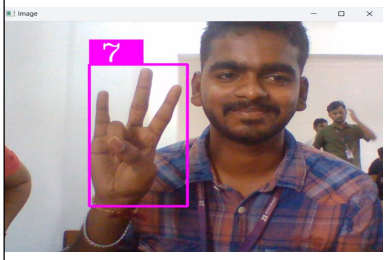
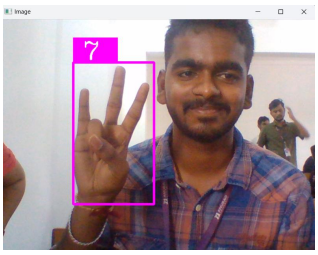
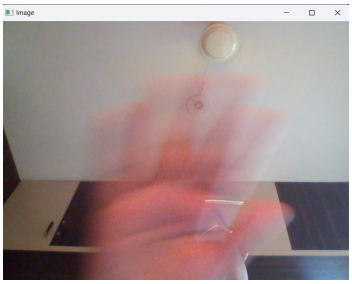
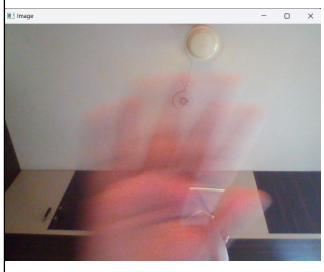
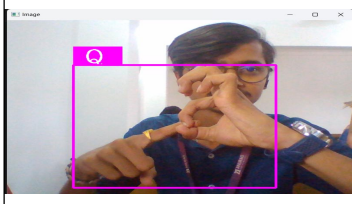
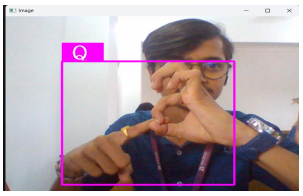
```
import cv2
from cvzone.HandTrackingModule import HandDetector
from cvzone.ClassificationModule import Classifier
import numpy as np
import math

cap = cv2.VideoCapture(0)
detector = HandDetector(maxHands=2)  # Detect up to 2 hands
classifier = Classifier("Model/keras_model.h5", "Model/labels.txt")

offset = 20
imgSize = 300

labels = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C", "D", "E", "F", "G", "H", "I",
"J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"]

while True:
    success, img = cap.read()
    imgOutput = img.copy()
    hands, img = detector.findHands(img)

    # Detect single-hand gestures
    if len(hands) == 1:
        hand = hands[0]
        x, y, w, h = hand['bbox']
        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255
        imgCrop = img[y - offset:y + h + offset, x - offset:x + w + offset]
```

```
        imgCropShape = imgCrop.shape
        aspectRatio = h / w

        if aspectRatio > 1:
            k = imgSize / h
            wCal = math.ceil(k * w)
            imgResize = cv2.resize(imgCrop, (wCal, imgSize))
            imgResizeShape = imgResize.shape
            wGap = math.ceil((imgSize - wCal) / 2)
            imgWhite[:, wGap:wCal + wGap] = imgResize
            prediction, index = classifier.getPrediction(imgWhite, draw=False)
            label = labels[index]
            confidence = prediction[index]
            print("Single-hand gesture:", label)

        else:
            k = imgSize / w
            hCal = math.ceil(k * h)
            imgResize = cv2.resize(imgCrop, (imgSize, hCal))
            imgResizeShape = imgResize.shape
            hGap = math.ceil((imgSize - hCal) / 2)
            imgWhite[hGap:hCal + hGap, :] = imgResize
            prediction, index = classifier.getPrediction(imgWhite, draw=False)
            label = labels[index]
            confidence = prediction[index]
            print("Single-hand gesture:", label)

        cv2.rectangle(imgOutput, (x - offset, y - offset - 50),
                      (x - offset + 90, y - offset - 50 + 50), (255, 0, 255), cv2.FILLED)
        cv2.putText(imgOutput, label, (x, y - 26), cv2.FONT_HERSHEY_COMPLEX, 1.7, (255,
255, 255), 2)
        cv2.rectangle(imgOutput, (x - offset, y - offset), (x + w + offset, y + h + offset), (255, 0, 255),
4)
```

```
# Detect double-hand gestures
elif len(hands) == 2:
    hand1 = hands[0]
    hand2 = hands[1]

    # Calculate the combined bounding box that includes both hands
    x1, y1, w1, h1 = hand1['bbox']
    x2, y2, w2, h2 = hand2['bbox']
    x = min(x1, x2)
    y = min(y1, y2)
    w = max(x1 + w1, x2 + w2) - x
    h = max(y1 + h1, y2 + h2) - y

    imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255
    imgCrop = img[y - offset:y + h + offset, x - offset:x + w + offset]

    imgCropShape = imgCrop.shape
    aspectRatio = h / w

    if aspectRatio > 1:
        k = imgSize / h
        wCal = math.ceil(k * w)
        imgResize = cv2.resize(imgCrop, (wCal, imgSize))
        imgResizeShape = imgResize.shape
        wGap = math.ceil((imgSize - wCal) / 2)
        imgWhite[:, wGap:wCal + wGap] = imgResize
        prediction, index = classifier.getPrediction(imgWhite, draw=False)
        label = labels[index]
        confidence = prediction[index]
        print("Double-hand gesture:", label)

    else:
        k = imgSize / w
        hCal = math.ceil(k * h)
```

```
            imgResize = cv2.resize(imgCrop, (imgSize, hCal))
            imgResizeShape = imgResize.shape
            hGap = math.ceil((imgSize - hCal) / 2)
            imgWhite[hGap:hCal + hGap, :] = imgResize
            prediction, index = classifier.getPrediction(imgWhite, draw=False)
            label = labels[index]
            confidence = prediction[index]
            print("Double-hand gesture:", label)

        cv2.rectangle(imgOutput, (x - offset, y - offset - 50),
                      (x - offset + 90, y - offset - 50 + 50), (255, 0, 255), cv2.FILLED)
        cv2.putText(imgOutput, label, (x, y - 26), cv2.FONT_HERSHEY_COMPLEX, 1.7, (255,
255, 255), 2)
        cv2.rectangle(imgOutput, (x - offset, y - offset), (x + w + offset, y + h + offset), (255, 0, 255),
4)

    cv2.imshow("Image", imgOutput)
    cv2.waitKey(1)
```

6. TEST CASES

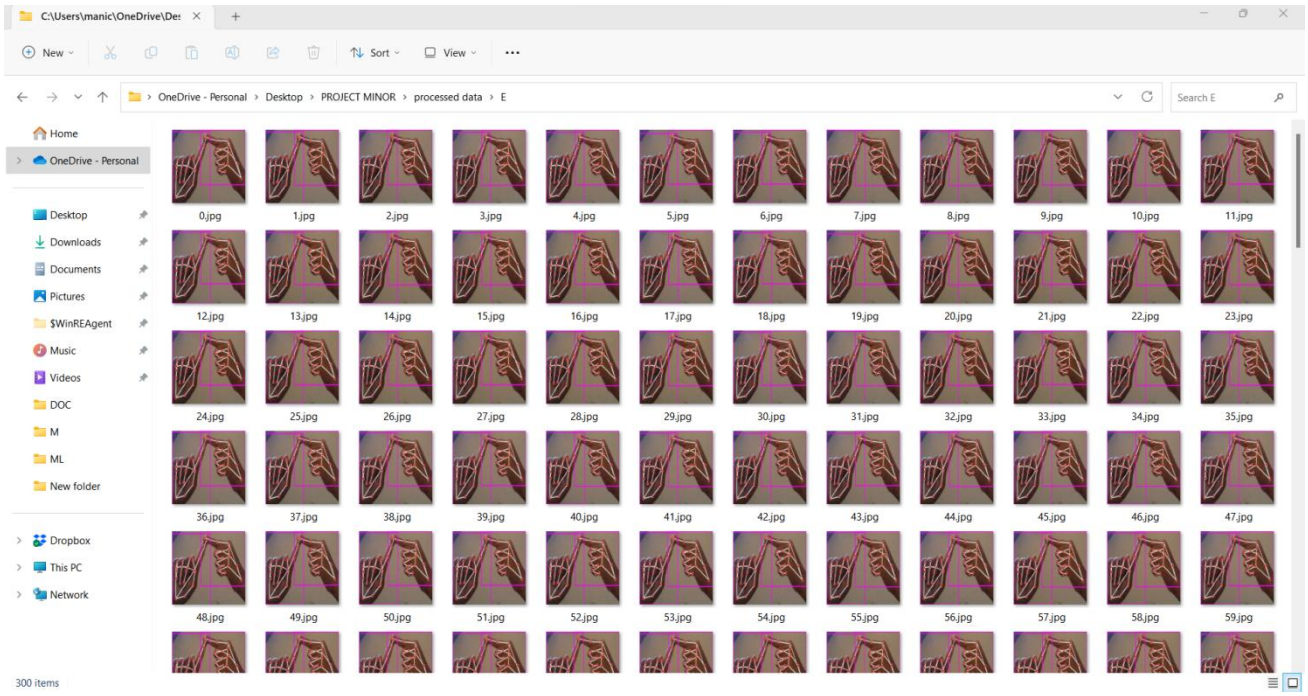
Test case ID	Input	Expected Output	Actual Output	Rate
1.	GESTURE 1 IS GIVEN			Success
2.	GESTURE H IS GIVEN			Success
3.	GESTURE 7 IS GIVEN			Success
4.	INTERRUPTION			Failure
5.	GESTURE Q IS GIVEN			Success

7.SCREEN SHOTS

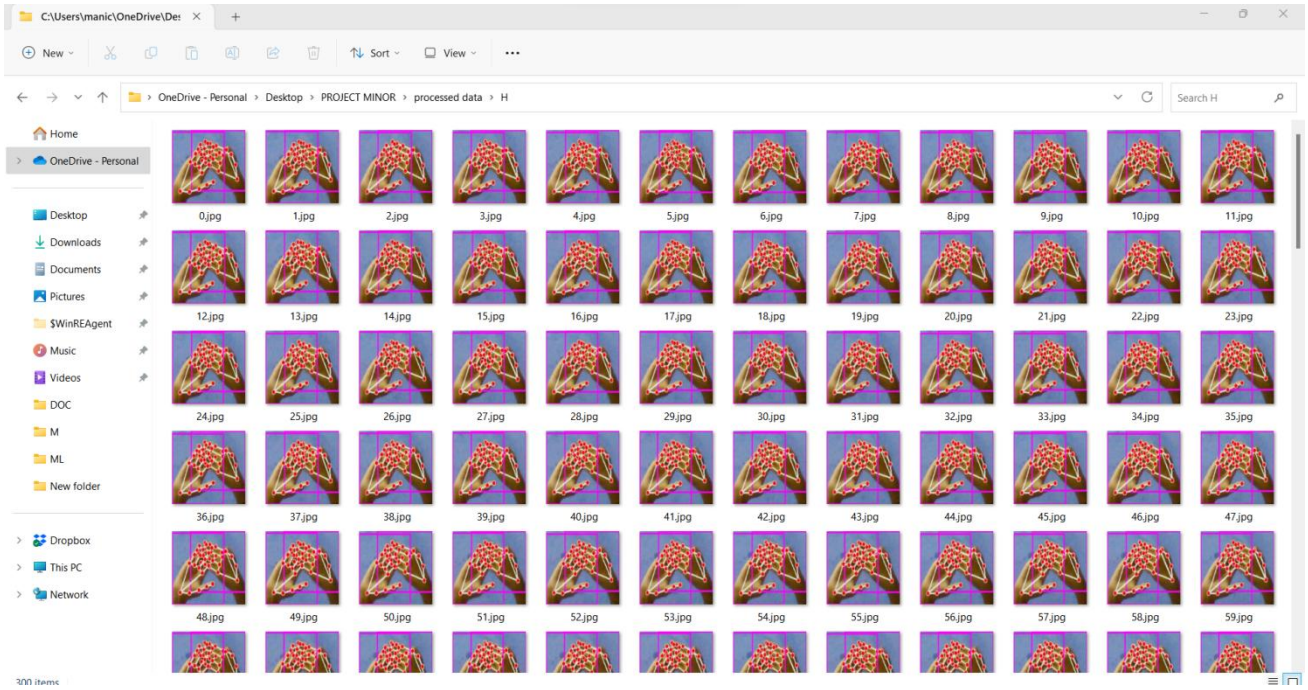
Datasets:

The development and availability of an Indian Sign Language dataset are pivotal for advancing research and applications related to sign language recognition and communication accessibility in India. Such a dataset would encompass a diverse range of signs, gestures, and expressions used within the culturally rich and linguistically diverse landscape of India's Deaf and Hard of Hearing community. It would serve as a crucial resource for training and testing machine learning models, enabling the creation of technology solutions like real-time sign language recognition systems, educational tools, and communication aids that cater to the unique linguistic nuances and regional variations in Indian Sign Language. Additionally, the dataset would contribute to the preservation and promotion of India's rich Deaf culture while fostering greater inclusivity and understanding between the Deaf and hearing populations.

Dataset Related To Alphabet 'E':



Dataset Related To Alphabet ‘H’:



TRAINING :

```
model.compile(loss = 'categorical_crossentropy',optimizer = 'adam',metrics = ["accuracy"])
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_12 (Conv2D)          (None, 222, 222, 32)      896

 max_pooling2d_12 (MaxPooli  (None, 111, 111, 32)      0
 ng2D)

 batch_normalization (Batch  (None, 111, 111, 32)      128
 Normalization)

 conv2d_13 (Conv2D)          (None, 109, 109, 64)      18496

 max_pooling2d_13 (MaxPooli  (None, 54, 54, 64)        0
 ng2D)

 batch_normalization_1 (Bat  (None, 54, 54, 64)        256
 chNormalization)

 conv2d_14 (Conv2D)          (None, 52, 52, 64)        36928

 max_pooling2d_14 (MaxPooli  (None, 26, 26, 64)        0
 ng2D)

 batch_normalization_2 (Bat  (None, 26, 26, 64)        256
 chNormalization)

 conv2d_15 (Conv2D)          (None, 24, 24, 96)        55392

 max_pooling2d_15 (MaxPooli  (None, 12, 12, 96)        0
 ng2D)

 batch_normalization_3 (Bat  (None, 12, 12, 96)        384
```

```
[ ]  epochs=5
     history = model.fit(train_data_gen,steps_per_epoch=int(np.ceil(8672/ float(32))),epochs=epochs,validation_data=val_data_gen,validation_steps=int(np.ceil(2168/ float(32))))
```

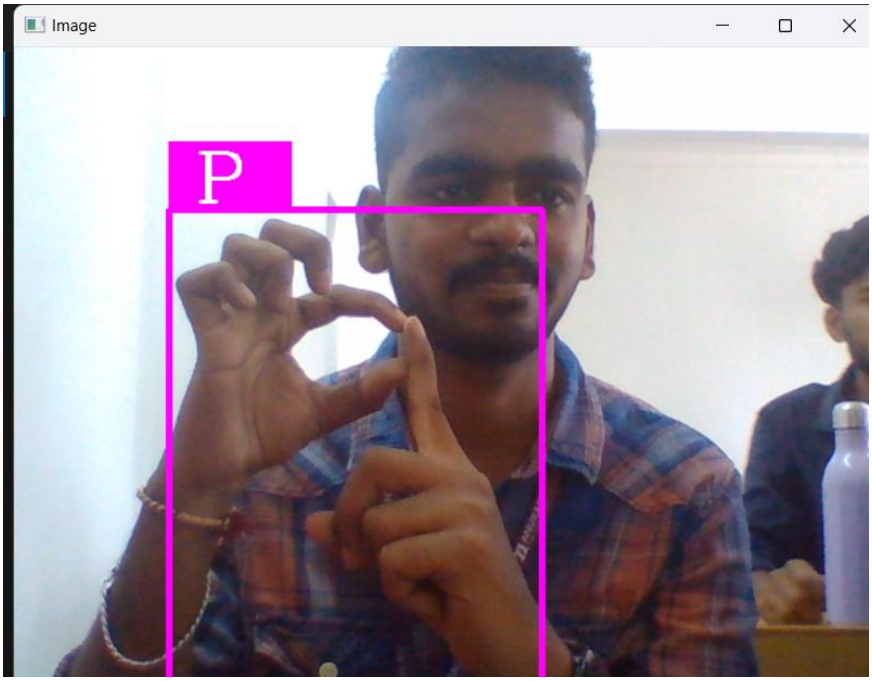
```
Epoch 1/5
271/271 [==============================] - 1220s 4s/step - loss: 1.1899 - accuracy: 0.6233 - val_loss: 9.0324 - val_accuracy: 0.0360
Epoch 2/5
271/271 [==============================] - 959s 4s/step - loss: 0.4038 - accuracy: 0.8577 - val_loss: 0.7252 - val_accuracy: 0.7652
Epoch 3/5
271/271 [==============================] - 949s 4s/step - loss: 0.2384 - accuracy: 0.9192 - val_loss: 0.5325 - val_accuracy: 0.8413
Epoch 4/5
271/271 [==============================] - 947s 3s/step - loss: 0.1745 - accuracy: 0.9415 - val_loss: 0.4684 - val_accuracy: 0.8722
Epoch 5/5
271/271 [==============================] - 941s 3s/step - loss: 0.1368 - accuracy: 0.9516 - val_loss: 0.1885 - val_accuracy: 0.9308
```

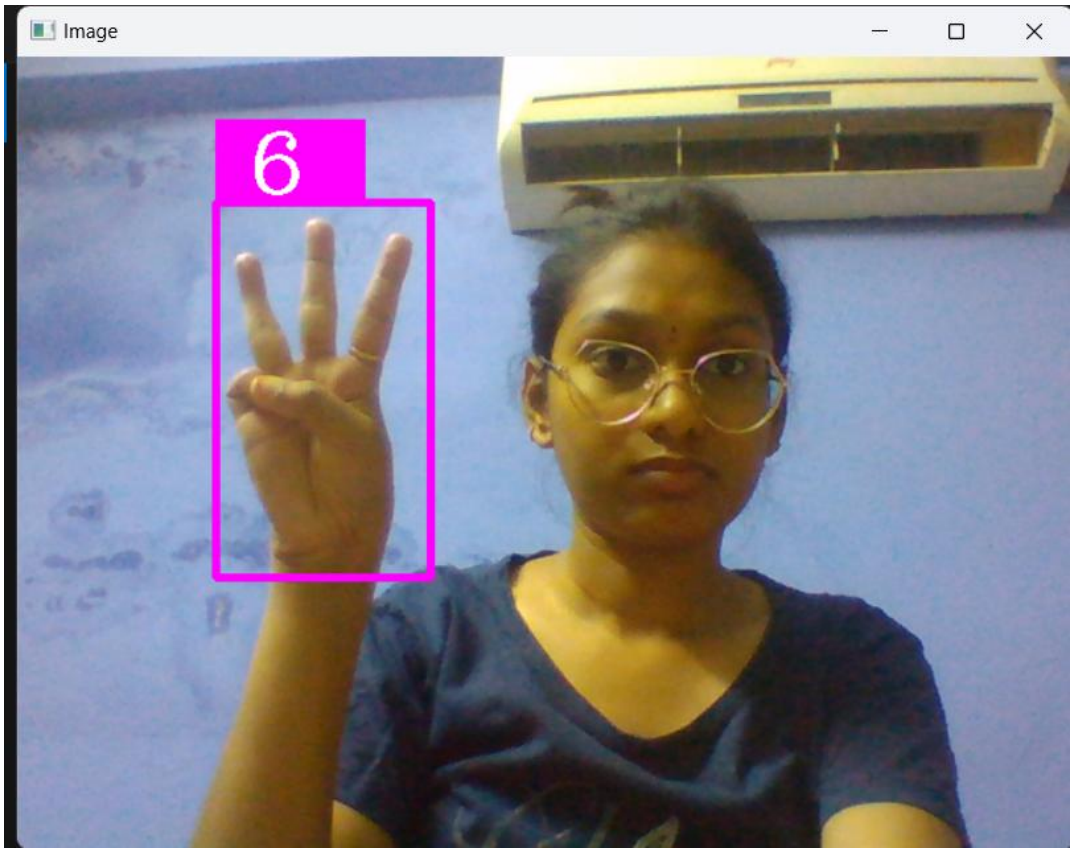
```
final_train_accuracy = history.history['accuracy'][-1]
print(f'Final training accuracy: {final_train_accuracy * 100:.2f}%')
```

```
Final training accuracy: 98.78%
```

```
[ ]  model.save('keras_model.h5')
```

TESTING OUTPUT:





8.CONCLUSION

In conclusion, real-time sign language recognition using Convolutional Neural Networks (CNN) represents a significant advancement in technology with the potential to greatly improve the lives of individuals with hearing impairments. CNN-based sign language recognition systems leverage the power of deep learning to accurately interpret and translate complex hand gestures into meaningful text or spoken language. By analyzing the visual features of sign language gestures in real-time video feeds, these systems provide a bridge for effective communication between the deaf and hearing communities. Moreover, the speed and efficiency of CNNs make real-time recognition feasible, allowing for immediate and natural interaction in various contexts, from educational settings to everyday conversations. As the field of computer vision and deep learning continues to evolve, real-time sign language recognition systems using CNNs hold promise for promoting inclusivity, breaking down communication barriers, and enhancing the quality of life for individuals with hearing impairments.

9.FUTURE ENHANCEMENTS

To enhance real-time sign language recognition with CNNs, it's essential to obtain larger, diverse datasets to improve generalization and transition from isolated sign recognition to continuous language understanding. Implementing 3D CNNs, real-time user feedback, multimodal sensing, and robust signer adaptation can enhance accuracy and usability. Expanding translation abilities, accommodating non-manual signals, and integration into real-world applications like video conferencing and customer support provide practical utility. Privacy and security must be ensured, and international sign language support broadens accessibility, ultimately making real-time sign language recognition more effective and inclusive for diverse users.

10.BIBLIOGRAPHY

1. J. Ekbote and M. Joshi, "Indian sign language recognition using ANN and SVM classifiers," 2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS), Coimbatore, India, 2017, pp. 1-5, doi: 10.1109/ICIIECS.2017.8276111.

2. M. Jaiswal, V. Sharma, A. Sharma, S. Saini and R. Tomar, "An Efficient Binarized Neural Network for Recognizing Two Hands Indian Sign Language Gestures in Real-time Environment," 2020 IEEE 17th India Council International Conference (INDICON), New Delhi, India, 2020, pp. 1-6, doi: 10.1109/INDICON49873.2020.9342454.

3. Raghuveera, T., Deepthi, R., Mangalashri, R. et al. A depth-based Indian Sign Language recognition using Microsoft Kinect. Sādhanā 45, 34 (2020). https://doi.org/10.1007/s12046-019-1250-6

4. Rastgoo, R., Kiani, K. & Escalera, S. Video-based isolated hand sign language recognition using a deep cascaded mode l. Multimed Tools Appl 79, 22965–22987 (2020). https://doi.org/10.1007/s11042-020-09048-5

5. Jiang, L., Huang, H., Dang, J., & Mo, Z. (2019). Real-time sign language recognition using an efficient CNN-LSTM model. In Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops (pp. 3039-3047).

6. Akbari, H., Zeng, K., Guo, Y., & Ye, H. (2018). Real-time American Sign Language recognition based on convolutional neural network. In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC) (pp. 2623-2628).

7. Starostenko, O., & Grosu, V. (2017). Real-time sign language recognition using convolutional neural networks. In Proceedings of the International Conference on Computer Vision (ICCV) Workshops (pp. 1586-1591).

8. Pugeault, N., Bodenhagen, L., & Krüger, N. (2014). Sign language recognition using a pre-processing step on depth data and CNN for the image classification step. In Proceedings of the International Conference on Computer Vision (ICCV) Workshops (pp. 556-562).