REAL TIME SIGN LANGUAGE RECOGNITION USING DEEP LEARNING

A PROJECT REPORT

Submitted By

PRIYADHARSINI P	210820205063
LAVANYA Y	210820205047
PIRATHIKSHA T	210820205055

In partial fulfillment for the award of the degree

Of

BACHELOR OF TECHNOLOGY

In

INFORMATION TECHNOLOGY

KINGS ENGINEERING COLLEGE, IRUNGATTUKOTAI

ANNA UNIVERSITY:CHENNAI 600025

April 2024

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that main project report **"REAL TIME SIGN LANGUAGE RECOGNITION USING DEEP LEARNING "** is a bonafide work of **"PRIYADHARSINI P, LAVANYA Y, PIRATHIKSHA T "** who carried out this main project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr.D.C.JULLIE JOSEPHINE BE.,ME.,ph.D.,	Ms.D.REJEES JENIFA B.E.,M.E.,
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Professor	**Assistant Professor**
Dept of Information Technology,	Dept of InformationTechnology
Kings Engineering College,	Kings Engineering College,
Irungattukottai,	Irungattukottai,
Chennai-602 117	Chennai-602 117.

ACKNOWLEDGEMENT

We thank God for his blessings and also for giving as good knowledge and strength in enabling us to finish our project. Our deep gratitude goes to our founder late **Dr.D. SELVARAJ,M.A.,M.Phil.,** for his patronage in the completion of our project. We like to take this opportunity to thank our honourable chairperson **Dr.S. NALINI SELVARAJ,M.COM.,MPhil.,Ph.D.** and honourable director, **MR.S. AMIRTHARAJ, M.Tech., M.B.A** for their support given to us to finish our project successfully. Also we would like to extend my sincere thanks to our respected Principal, **Dr.C. RAMESH BABU DURAI, M.E.,Ph.D.** for having provided me with all the necessary facilities to undertake this project.

We are extremely grateful and thanks to our Head of the Department **Dr.D.C. JULLIE JOSEPHINE B.E.,M.E.,ph.D.,** for her valuable suggestion, guidance and encouragement. We wish to express our sense of gratitude to our project guide **Ms.D REJEES JENIFA B.E.,M.E.,** Assistant Professor of Information Technology Department, Kings Engineering College with his guidance and direction made our project a grand success. We express our sincere thanks to our parents, friends and staff members, who have helped and encouraged us during the entire course of completing this project work successfully.

PROBLEM STATEMENT:

The communication gap between deaf and hearing communities presents significant challenges, as sign language, the primary communication mode for many deaf individuals, is not widely understood. Current translation technologies fail to capture the nuances and dynamism of sign language, lacking real-time, universally accessible translation capabilities. The proposed sign language recognition system aims to address this by harnessing advanced machine learning and computer vision technologies. The system will capture and interpret sign language through a camera, using a Convolutional Neural Network (CNN) to translate gestures into text and speech. A critical aspect of the development involves an algorithm sophisticated enough to accurately process the varied expressions of sign language, catering to individual differences in gesture execution.

The end goal is to deploy the system effectively in real-time scenarios, ensuring it performs reliably under diverse conditions. After rigorous training on a diverse dataset, the system will be tested in real-world environments, fine-tuning its ability to provide instantaneous translations. This capability will extend beyond one-on-one interactions, potentially revolutionizing communication in educational, professional, and casual settings. Ultimately, this innovation aims to dismantle barriers, enhance service accessibility, and foster inclusivity, creating a world where communication with the deaf community is fluid and unrestricted by language constraints.

ABSTRACT

One of the most natural and ancient types of conversational language is sign language. The technology that converts sign language into writing for people who have difficulty communicating, such as those who have speech issues, hearing disabilities, or are deaf, is the subject of this study. This paper is based on a real-time method based on finger writing and neural networks for American sign language. An interesting field of vision study is the automatic recognition of human gestures from video images. The Research recommend employing a convolution neural network (CNN) method to recognize human hand motions from a photograph. The objective is to identify hand movements used in human work activities from a camera image. Hand placement and orientation are employed to collect the training and assessment data for CNN. The hand is first put through a filter, and once that has been done, it is put through a classification, which determines what class the hand movements belong to. Then, CNN is trained using the measured pictures.

LIST OF FIGURES

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

American sign language is a predominant sign language Since the only disability D&M people have been communication related and they cannot use spoken languages hence the only way for them to communicate is through sign language. Communication is the process of exchange of thoughts and messages in various ways such as speech, signals, behaviour and visuals. Deaf and dumb(D&M) people make use of their hands to express different gestures to express their ideas with other people. Gestures are the nonverbally exchanged messages and these gestures are understood with vision. This nonverbal communication of deaf and dumb people is called sign language. In our project we basically focus on producing a model which can recognise hand gestures in order to form a complete word by combining each gesture. The gestures we aim to train are as given in the image below.

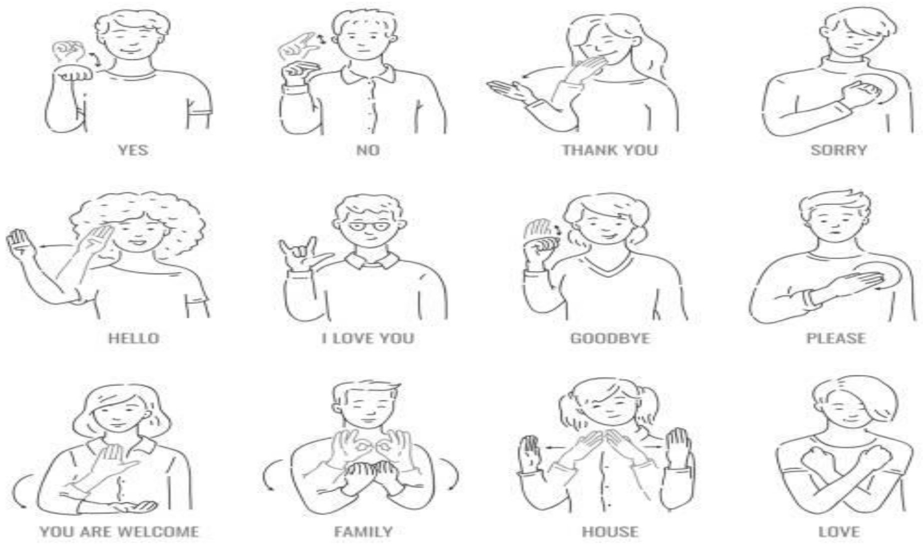


Figure 1.1: American Sign Language

1.1 AN OVERVIEW

1.1.1 Deep learning

Deep learning is an artificial intelligence function that imitates the workings of the human brain in processing data for use in detecting objects, recognizing speech, translating languages, and making decisions. Deep learning is a subfield of machine learning where algorithms inspired by the structure and function of the brain, called artificial neural networks, learn from large amounts of data. Similar to how we learn from experience, the deep learning algorithm would perform a task repeatedly, each time tweaking it a little to improve the outcome.

A deep learning model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labeled data and neural network architectures that contain many layers.

1.1.2 Key Components of Deep Learning

Neural Networks:

At its core, deep learning uses neural networks. These networks are composed of interconnected nodes or neurons arranged in layers. The input layer receives the data, hidden layers process the data, and the output layer produces the prediction or result.

Layers:

Deep learning gets its name from the number of hidden layers in the neural network. Traditional neural networks might contain 2-3 hidden layers, while deep networks can have as many as 150.

Weights and Biases:

Connections between neurons have associated weights, and neurons have biases. During the learning process, the network adjusts these weights and biases to minimize the difference between its prediction and the actual data.

Activation Functions: These functions determine the output of a neural network node, or neuron, and add non-linear properties to the system, allowing it to learn more complex tasks.

Forward Propagation:

Data moves through the neural network, from input to output layers, and the results are analyzed.

Backpropagation:

After the output is compared with the actual result, the error is sent back through the network to adjust and improve the weights. This process is typically done using algorithms like Gradient Descent.

Loss Functions:

These functions are used to measure how well the model predicts the target variable. Common loss functions include mean squared error for regression tasks and cross-entropy for classification tasks.

Optimizer:

This is an algorithm or method used to change the attributes of the neural network, such as weights and learning rate, to reduce the losses.

Epochs:

Training occurs over epochs. An epoch is one complete presentation of the data set to be learned to a learning machine.

Automated Driving:

Deep learning is used to detect objects like stop signs and traffic lights. In addition, deep learning is used to detect pedestrians, which helps decrease accidents.

Aerospace and Defense:

Deep learning is used to identify objects from satellites that locate areas of interest and safe or unsafe zones for troops.

Medical Research:

Cancer researchers are using deep learning to automatically detect cancer cells.

Industrial Automation:

Deep learning is improving worker safety around heavy machinery by automatically detecting when people or objects are within an unsafe distance of machines.

Electronics:

Deep learning is being used in automated hearing and speech translation. For example, home assistance devices like Alexa and Google Home use deep learning to understand speech and language of users.

Overall, deep learning is a powerful AI technique that's pushing the boundaries of what machines can do and will continue to be a crucial technology for achieving artificial general intelligence.

1.1.3 Fundamentals of Deep Learning

Deep learning models are designed to learn in a manner that is akin to human learning, although at a more abstract and complex level. These models are composed of multiple processing layers (hence "deep") to learn representations of data with multiple levels of abstraction. Through the use of a vast amount of data and computational power, these models can automatically discover the representations needed for feature detection or classification, rather than relying on human-engineered features.

1.1.4 Architecture of Deep Neural Networks

At the heart of deep learning is the neural network architecture. Though inspired by the biological neural networks that constitute animal brains,

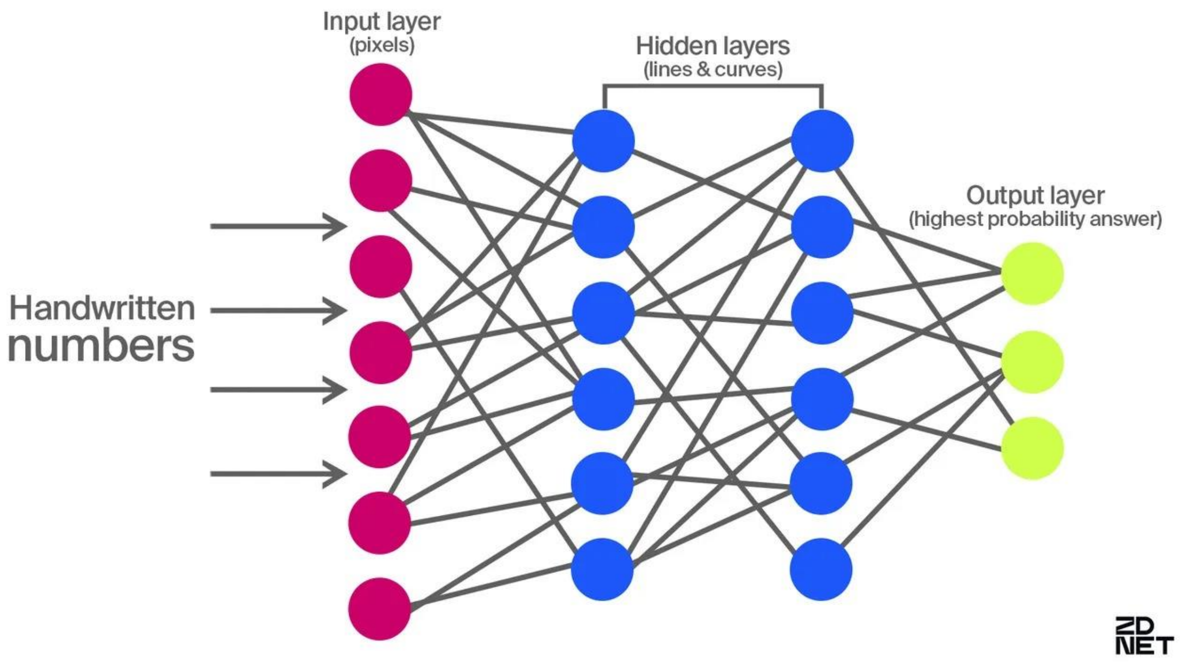


Figure 1.2 : Architecture of Deep Neural Networks

artificial neural networks (ANNs) are simplified and structured in a way that enables clear numerical computation. These networks are composed of layers of interconnected nodes:

Input Layer:

This is where the network receives its input data.

Hidden Layers:

These layers perform various computations through a system of weighted connections. The more hidden layers, the "deeper" the network, and the more complex the functions it can learn.

Output Layer:

This layer outputs the final prediction or decision of the network.

1.1.5 Challenges and Solutions

One of the critical challenges in deep learning is overfitting, where the model performs well on the training data but poorly on unseen data. Techniques like dropout, data augmentation, and regularization are employed to combat overfitting and improve the model's generalization.

1.1.6 Advanced Deep Learning Concepts

Convolutional Neural Networks (CNNs):

These are specialized kinds of neural networks for processing data that have a known grid-like topology, such as images.

Recurrent Neural Networks (RNNs):

These networks are designed to recognize sequences, such as time series or language, by processing the sequence one element at a time while maintaining a memory (state) of what has been processed.

Generative Adversarial Networks (GANS):

These are neural architectures where two networks, typically a generator and a discriminator, are trained simultaneously in a zero-sum game framework.

Real-World Applications

1.1.7 The applications of deep learning are vast and ever-growing

Autonomous Vehicles:

Deep learning models process inputs from various sensors and cameras to make decisions, enabling vehicles to navigate safely.

Healthcare:

From diagnostics to drug discovery, deep learning is helping to predict patient outcomes, diagnose diseases, and personalize treatment.

Natural Language Processing (NLP):

Deep learning is at the forefront of translating languages, generating text, and understanding speech with applications in virtual assistants, chatbots, and more.

Finance:

In finance, deep learning algorithms are used for fraud detection, portfolio management, algorithmic trading, and risk assessment.

1.1.8 The Future of Deep Learning

As datasets continue to expand and computational capabilities become more widely available, the potential of deep learning only grows. One of the most anticipated developments in AI, fueled by deep learning, is the advent of artificial general intelligence (AGI). AGI represents a type of intelligence that could perform any intellectual task that a human being can. Unlike current AI systems that excel at specific tasks, AGI would have the ability to understand, learn, and apply knowledge in a wide variety of disciplines, essentially mirroring the cognitive abilities of humans

The implications of such an advancement are profound, potentially heralding a new era in which machines could assist or even lead innovations across sectors like healthcare, finance, education, and more. For instance, AGI could enable the development of highly personalized medical treatments by synthesizing information from genomics, medical imaging, and patient history. It could revolutionize education by providing tailored learning programs that adapt to each student's unique needs and learning style.

However, the evolution of deep learning and the pursuit of AGI also introduce significant ethical and societal challenges. As AI systems become more capable, concerns about data privacy, security, and the ethical use of technology are amplified. The ability of deep learning algorithms to make decisions based on large datasets raises questions about the biases those datasets may contain. Moreover, the potential misuse of AI technologies, such as in autonomous weapons or surveillance systems, poses risks that must be managed through careful governance and regulation.

Ensuring that the development of deep learning and AGI benefits humanity requires a collaborative approach among researchers, policymakers, industry leaders, and ethicists. Together, they must navigate the balance between innovation and the potential risks associated with these powerful technologies. As deep learning continues to shape the future of AI, the focus on responsible and ethical AI practices becomes not just a recommendation but a necessity for the well-being and security

1.2 OBJECTIVE

The objective of developing sign language recognition based on hand symbol's classification is to create a technology that can accurately interpret and understand sign language gestures performed by individuals who are deaf or hard of hearing. The goal is to bridge the communication gap between sign language users and non-sign language users, enabling better accessibility and inclusivity for the deaf community.

By developing a system that can recognize and interpret sign language gestures, it becomes possible to convert sign language into written or spoken language, allowing sign language users to communicate more effectively with those who do not understand sign language. This technology can be applied in various domains, such as education, communication, and accessibility, providing new opportunities for individuals with hearing impairments.

The development of sign language recognition systems involves training machine learning models to analyze and classify the hand symbols used in sign language. These models can utilize techniques such as computer vision, deep learning, and pattern recognition to accurately identify and interpret the gestures. The objective is to achieve high accuracy and real-time recognition, enabling seamless communication between sign language users and the wider population.

1.3 PROJECT SCOPE

This System will be Beneficial for Both Dumb/Deaf People and the People Who do not understands the Sign Language. They just need to do that with sign Language gestures and this system will identify what he/she is trying to say after identification it gives the output in the form of Text as well as Speech form.

The sign language recognition system endeavors to achieve both high accuracy and robustness, capable of accurately identifying hand gestures across diverse environments, lighting conditions, and subjects while remaining resilient to variations in gesture appearance and background noise. Real-time performance is paramount, necessitating the system's ability to process video streams swiftly and with minimal latency, providing immediate feedback on recognized gestures. This requires efficient optimization of model inference and image processing pipelines to ensure seamless interaction. Additionally, the system prioritizes user experience by creating an intuitive and responsive interface that leverages hand gesture recognition to facilitate natural and efficient interactions. Accessibility and ease of use are emphasized to accommodate users of varying technological proficiency, promoting inclusivity and enhancing overall usability.

1.4 CHALLENGES

Data Diversity:

Acquiring a diverse and representative dataset is essential for training a robust sign language recognition system. This involves sourcing data encompassing various sign language gestures, hand shapes, movements, and orientations. Each gesture in the dataset must be accurately annotated with corresponding labels, ensuring the model learns to recognize different signs effectively. Challenges may arise in obtaining sufficient data for less common signs or variations in sign language across different regions and communities. Collaborating with sign language experts and communities can aid in curating a comprehensive dataset that adequately represents the diversity of sign language gestures.

Model Complexity:

Designing an effective CNN model entails finding the right balance between complexity and generalization. A model that is too simple may struggle to capture the intricate patterns present in sign language gestures, leading to poor recognition performance. Conversely, an overly complex model risks overfitting to the training data, failing to generalize well to unseen examples. Achieving the optimal model architecture involves experimentation with different network architectures, layer configurations, and regularization techniques. Techniques such as dropout and batch normalization may be employed to prevent overfitting and improve generalization capabilities.

Generalization:

Ensuring the sign language recognition model generalizes well to unseen data is crucial for its real-world applicability. Variability in hand shapes, movements, backgrounds, and lighting conditions can pose challenges to the model's ability to accurately recognize signs in different environments. Data augmentation techniques, such as random rotations, translations, and changes in lighting conditions, can help expose the model to diverse examples during training, enhancing its robustness to variations. Additionally, transfer learning from pre-trained models on large-scale datasets may aid in improving generalization by leveraging features learned from related tasks.

Real-time Processing:

Achieving real-time performance in sign language recognition requires optimizing the model inference process to minimize latency. This involves optimizing the computational efficiency of the model, including reducing the number of parameters, optimizing network architecture for inference speed, and employing hardware acceleration techniques such as GPU or specialized inference chips. Furthermore, efficient image preprocessing techniques, such as resizing, cropping, and normalization, can streamline the input data pipeline, enabling faster inference without compromising recognition accuracy. Implementing efficient algorithms and data structures for model deployment on resource-constrained devices, such as smartphones or embedded systems, is also essential for real-time processing in practical applications.

Speech Synthesis:

Converting recognized sign language gestures into spoken language involves employing speech synthesis techniques to generate natural-sounding speech. This requires developing algorithms that can transform textual representations of recognized signs into fluent and intelligible speech output. Techniques such as concatenative synthesis, where pre-recorded speech segments are concatenated to form complete utterances, or parametric synthesis, where speech is generated based on learned acoustic models, may be utilized. Additionally, incorporating prosody and intonation patterns characteristic of spoken language can enhance the naturalness and expressiveness of synthesized speech.

User Interaction:

Designing an intuitive and responsive user interface is essential for facilitating natural interaction with the sign language recognition system. The interface should provide clear visual feedback on recognized signs and synthesized speech output, enabling users to communicate effectively. Incorporating user feedback mechanisms, such as gesture corrections or alternative input methods, can improve the system's accuracy and user satisfaction over time. Additionally, ensuring accessibility features, such as support for alternative input devices or customization options for user preferences, enhances inclusivity and usability for individuals with diverse needs. Regular usability testing and iterative refinement of the user interface based on user feedback are essential for optimizing the user experience and ensuring the system meets the needs of its target audience.

CHAPTER 2
LITERATURE SURVEY

2.1 Image-based and sensor-based approaches to American sign language recognition

Sign language is a unique communication tool helping to bridge the gap between people with hearing impairments and the general public. It holds paramount importance for various communities, as it allows individuals with hearing difficulties to communicate effectively. In sign languages, there are numerous signs, each characterized by differences in hand shapes, hand positions, motions, facial expressions, and body parts used to convey specific meanings. The complexity of visual sign language recognition poses a significant challenge in the computer vision research area. This study presents an American Sign Language recognition (AmSL) system that utilizes convolutional neural networks (CNNs) and several transfer learning models to automatically and accurately identify American Sign Language characters. The dataset used for this study comprises 54,049 images of AmSL words. The results of this research indicate that InceptionV3 outperformed other pretrained models, achieving a remarkable 100% accuracy score and a 0.00 loss score without overfitting. These impressive performance measures highlight the distinct capabilities of InceptionV3 in recognizing American characters and underscore its robustness against overfitting. This enhances its potential for future research in the field of American Sign Language recognition.

2.2 Recognition of sign language words and numbers based on hand kinematics using a data glove

This paper reports real-time recognition of Indian and American sign language words and numbers based on hand kinematics assessment. The finger and wrist joint angles were acquired using an indigenously developed data glove. The data set was for single handed Indian sign language words American sign language words and sign numbers. The data were pre-processed through a moving average filter and standardized feature scaling methods. The glove was able to measure the finger joint angles with an accuracy standard deviation for metacarpophalangeal (MCP) joint 2.14°, proximal inter phalangeal (PIP) joint± 1.73° and distal inter phalangeal (DIP) joint 1.49°, during flexion/extension and abduction/adduction movements. A radial basis function kernel support vector machine with 10-fold cross validation was used for recognition. An average recognition rate of 96.7% was achieved. Using a label matching and speech data base, the recognising words and numbers were translated into speech

2.3 Nearest neighbour classification of Indian sign language gestures using kinect camera

People with speech disabilities communicate in sign language and therefore have trouble in mingling with the able-bodied. There is a need for an interpretation system which could act as a bridge between them and those who do not know their sign language. A functional unobtrusive Indian sign language recognition system was implemented and tested on real world data. A vocabulary of 140 symbols was collected using 18 subjects, totalling 5041 images. The vocabulary consisted mostly of two-handed signs which were drawn from a wide repertoire of words of technical and daily-use origins. The system was implemented using Microsoft Kinect which enables surrounding light conditions and object colour to have negligible effect on the efficiency of the system. The system proposes a method for a novel, low-cost and easy-to-use application, for Indian Sign Language recognition, using the Microsoft Kinect camera. In the fingerspelling category of our dataset, we achieved above 90% recognition rates for 13 signs and 100% recognition for 3 signs with overall 16 distinct words recognised with an average accuracy rate of 90.68%.

2.4 Sign Language Recognition Using Deep Learning on Custom Processed Static Gesture Images

Sign Language detection by technology is an overlooked concept despite there being a large social group which could benefit by it. There are not many technologies which help in connecting this social group to the rest of the world. Understanding sign language is one of the primary enablers in helping users of sign language communicate with the rest of the society. Convolutional neural networks have been employed in this paper to recognize sign language gestures. The image dataset used consists of static sign language gestures captured on an RGB camera. Preprocessing was performed on the images, which then served as the cleaned input. The paper presents results obtained by retraining and testing this sign language gestures dataset on a convolutional neural network model using Inception v3. The model consists of multiple convolution filter inputs that are processed on the same input. The validation accuracy obtained was above 90% This paper also reviews the various attempts that have been made at sign language detection using machine learning and depth data of images. It takes stock of the various challenges posed in tackling such a problem, and outlines future scope as well.

2.5 A new data glove approach for Malaysian sign language detection

A normal human being sees, listens, and reacts to his/her surroundings. There are some individuals who do not have this important blessing. Such individuals, mainly deaf and dumb, depend on communication via sign language to interact with others. Furthermore, this will cause a problem for the deaf and dumb communities to interact with others, particularly when they attempt to involve with educational, social and work environments. In this research, the objectives are to develop a sign language translation system in order to assist the hearing or speech impaired people to communicate with normal people, and also to test the accuracy of the system in interpreting the sign language. As a first step, the best method in gesture recognition was chosen after reviewing previous researches. The configuration of the data glove includes 10 tilt sensors to capture the finger flexion, an accelerometer for recognizing the motion of the hand, a microcontroller and Bluetooth module to send the interpreted information to a mobile phone. Firstly the performance of the tilt sensor was tested. Then after assembling all connections, the accuracy of the data glove in translating some selected alphabets, numbers and words from Malaysian Sign Language is performed. The result for the first experiment shows that tilt sensor need to be tilted more than 85 degree to successfully change the digital state. For the accuracy of 4 individuals who tested this device, total average accuracy for translating alphabets is 95%, numbers is 93.33% and gestures is 78.33%. The average accuracy of data glove for translating all type of gestures is 89%. This fusion of tilt sensors and accelerometer could be improved in the future by adding more training and test data as well as underlying frameworks such as Hidden Markov Model.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING METHOD

Contemporary sign language recognition systems are marred by suboptimal accuracy, particularly in varied lighting and backgrounds, leading to an unsatisfactory user experience. The issue is compounded by considerable delays in sign interpretation, which impedes real-time communication—a critical component for effective sign language translation. These systems often do not generalize well to the broad spectrum of sign language expression due to limited training datasets that fail to encapsulate the diversity of the signing population. Moreover, reliance on specialized hardware such as sensor-equipped gloves adds an economic and practical barrier to technology adoption, limiting the potential for widespread usage.

3.2 PROPOSED METHEDOLOGY

The envisioned methodology augments Convolutional Neural Networks (CNNs) with advanced feature extraction techniques to revolutionize sign language recognition. A well-architected CNN, equipped with multiple specialized layers and strategic dropout incorporation, is essential for nuanced feature detection and generalization. The training regimen is thorough, emphasizing validation to confirm the model's accuracy and its ability to generalize. Real-time processing capabilities are prioritized, aiming for a seamless, immediate translation of sign language gestures. The system's design targets intuitive user interaction, minimizing latency to enable natural communication flows, and negates the need for restrictive hardware, thereby increasing the accessibility and adaptability of the technology across various user scenarios.

Year	**Ref**	**Methodology**	**Accuracy**
2021	[23]	Convolutional neural networks (CNNs) and transfer learning models, dataset of 54,049 AmSL images, with InceptionV3	98%
2021	[14]	SVM with 10-fold cross-validation	96.7%
2020	[22]	A Kinect camera capturing 5041 images across 140 symbols, recognition rates	90%
2022	[6]	Deep learning with CNNs on processed static gesture images using Inception v3	90%
2021	[21]	Development of a data glove with tilt sensors and accelerometer	89%
2024	Proposed System	Convolutional neural networks (CNN)+Feature extraction (PCA, LBP, HoG), MediaPipe	100%

Figure 3.1 :Compare with another methodology with accuracy

CHAPTER 4

SYSTEM REQUIREMENTS

4.1 HARDWARE REQUIREMENTS :

• Hard Disk	: 500GB and above
• RAM	: 4GB and above
• Processor	: I5 and above

4.2 SOFTWARE REQUIREMENTS :

• Software	: Python
• Tools	: Anaconda (spyder python)
• Operating System	: Windows 7,8,10 (64 bit)

4.3 REQUIRED LIBRARIES :

TensorFlow: An open-source library for machine learning, utilizing data flow graphs for building models.

Keras: A Python-based neural networks API that runs on top of TensorFlow, designed for fast experimentation.

NumPy: A Python package for scientific computing, supporting large, multi-dimensional arrays and matrices.

OpenCV (cv2): An open-source library for computer vision and machine learning, enhancing machine perception applications.

MediaPipe: A framework for creating multimodal machine learning pipelines, emphasizing real-time and cross-platform applications.

CHAPTER 5

SYSTEM ARCHITECTURE

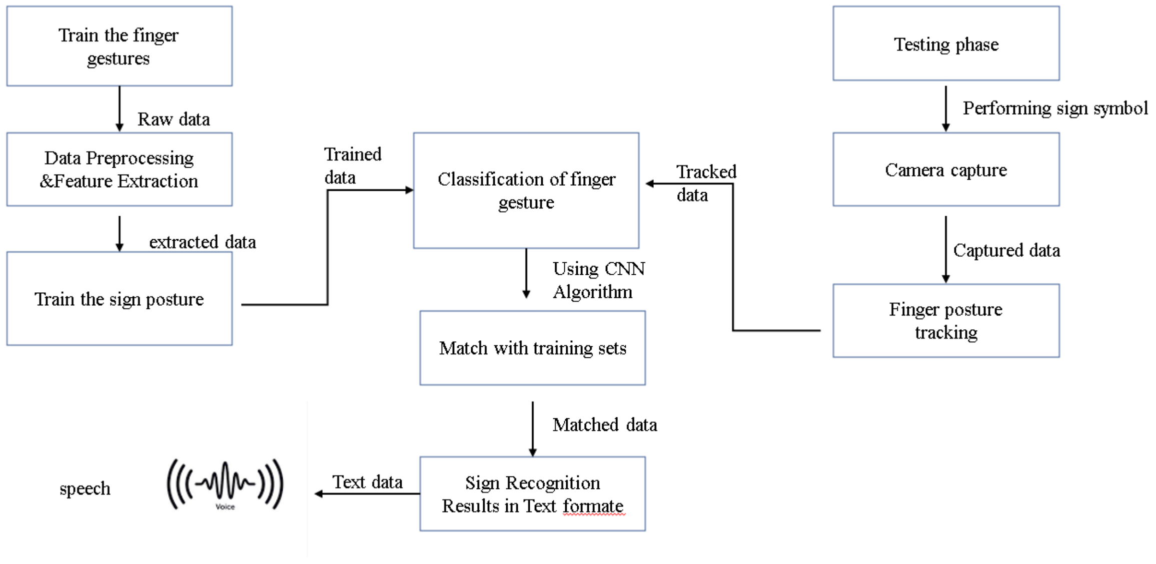


Figure 5.1 :System Diagram

The system architecture for sign language recognition based on hand symbols typically involves multiple components working together to achieve accurate and real-time recognition. Here is a suggested system architecture:

Training Phase:Train the Finger Gestures: In the initial training phase, the system is exposed to a vast array of finger gestures representing different signs. This exposure consists of capturing raw data, likely through images and videos, which showcase the various hand shapes and movements used in sign language.

Data Preprocessing: Raw data typically includes a lot of irrelevant information or noise. The preprocessing step aims to clean and transform this raw data into a format that's more suitable for machine learning. Techniques used could include image resizing, normalization (scaling pixel values), color space transformations, and potentially augmenting the data set to include a broader variety of gestures under different conditions to improve the robustness of the model.

Train the Sign Posture: With preprocessed data, the next step involves using machine learning algorithms to train a model to recognize different sign postures. This stage will use the refined data to train a CNN, allowing the network to learn features that are important for recognizing and differentiating between various sign language gestures. The training involves adjusting the weights of the network through a series of iterations to minimize the error in predictions, using methods such as backpropagation.

Classification Phase:Classification of Finger Gesture: After training, the system uses the CNN to classify finger gestures. The CNN, known for its ability to identify hierarchical patterns in images, processes the input to determine the specific gesture being made.

Match with Training Sets: This classification isn't done in isolation; the CNN compares the features of the new input with the trained data to find the closest match. The trained dataset acts as a reference library, and the CNN uses this library to label new data based on learned patterns.

Testing Phase:Performing Sign Symbol: This phase involves real-world application. Individuals perform sign symbols in front of a camera, acting as a live test for the system to demonstrate its ability to recognize signs in various conditions and potentially by different signers.

Camera Capture: A camera records the sign performance, translating the analog movements into digital data that the system can analyze.

Finger Posture Tracking: With sophisticated tracking algorithms, the system follows the specific movements and positioning of the fingers as captured in the video feed. The real-time processing compares the captured gestures against the model's learned dataset to predict and understand the sign being communicated.

Output Phase:Sign Recognition Results in Text Format: The successful recognition of a sign gesture is then converted into its corresponding text format. This text representation can be displayed to the user or stored for further processing, enabling a written record of the sign language communication.

Speech: To further enhance accessibility, the system likely includes a text-to-speech module that can articulate the recognized text aloud. This conversion bridges the communication gap, allowing hearing individuals to understand the translation of the sign language in real time.

CHAPTER 6
REVIEW METHOD

In this study, we summarize and organize scientific data about the subject of Sign Language Recognition (SLR) for the benefit of the entire research community. In order to assist anyone interested in the fundamental knowledge in this field, we complemented the basic facts about each studywith an impartial assessment of its quality and potential for positive contributions.

We attempt to answer the following main research questions:

Question 1 – Which studies have been conducted addressing automated Sign Language Recognition, and what are the available datasets?
Question 2– What techniques in Automatic Sign Language Recognition for various languages are applied to date?
Question 3 – Which challenges remain unsolved in thisscientific field?

One of the ultimate objectives of this paper is to lay the groundwork for future inquiries about SLR and clarify any ambiguous elements that might confuse some researchers. We accomplished this in three phases – preparatory, execution and presentation, with each stage including several steps. These steps included 1) selecting the most relevant research questions, 2) setting fundamental rules for the eval- uation procedure, 3) formalizing the selection threshold, 4) assessing the quality of the work's premises and results, 4) looking into the methodological setup of the experiments, and 5) extracting any bits of information that contribute to answering the central questions.

6.1 REVIEW PROTOCOL

We followed a defined procedure during the literature review, allowing for a more objective evaluation of the paper content. This procedure consisted of numerous tasks, starting with selecting relevant variables, isolating the authors' strategic approaches, analyzing the methods and techniques used to obtain the results, sorting out quantitative output, and defin-ing the principles for generalization and summarization.

6.1.1 INCLUSION AND EXCLUSION CRITERIA

parameters were used to decide which works to include. Since the subject of this paper is SLR, only papers from this field were taken into consideration. The period covered is between 2014 and 2021, as shown in Figure, as the idea was to provide a systematization of contemporary research. In the following figure , we provide a complete set of rules for selecting the research papers in a succinct format.

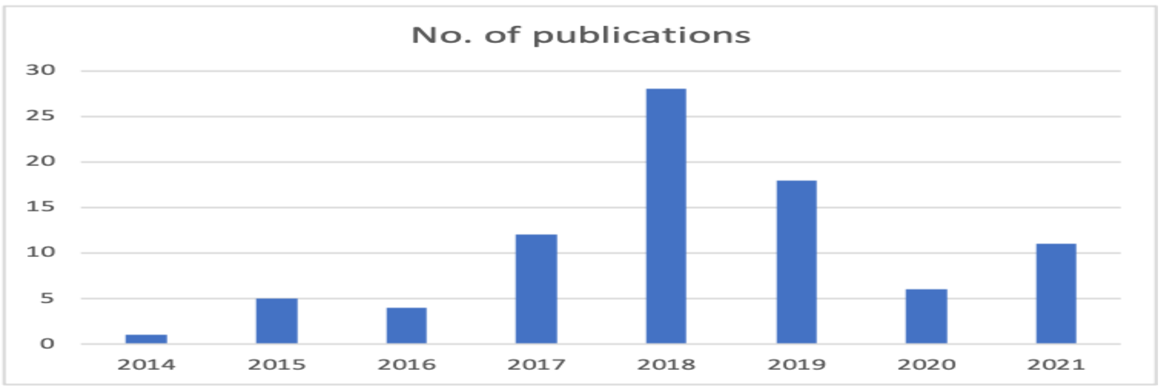


Figure 6.1: Number of publications on Sign Language Recognition by year

6.2 SEARCH STRATEGY

Finding the most relevant research material required an arduous process combing publicly available sources usinga combination of automated tools and human workforce. Specific keywords drove the automated segment, which are display.

This collection of studies is continually expanded through addition of individual papers that match the same level of relevance as those found by the algorithm. We included all of the most significant online repositories of scientific content in the search, from Google Scholar, MDPI, Springer, Elsevier, and IEEE explore to ACM and ArXiv. The proportion of papers from each source.

The overall objective at this stage was to discover as many works that address the topic of SLR as possible. After completing this stage, we carefully analyzed the entire corpus of collected material using the forward/back technique. Thisallowed for a more detailed understanding of each paper, with the ability to track all references and follow the significant lines of research. In this way, it was possible to ensure that no foundational studies are missing from the study and that the final collection of SLR papers is truly representative of the most successful research directions.We then processed the collection based on the Mendeley method, which madeit possible to easily identify and remove identical items from the list, making the content more readily searchable. We noted several trends in this part of the process, which included a breakdown of collected works based on the local variation of sign language they refer to. A majority of works in the collection (more than 30%) were related to the American variation, but French, Argentinian, Arabic, and many other SL variations are also represented, as seen. Another factor that was used to differentiate between papers is the type of architecture of the proposed solution. Fulloverview is.

6.3 STUDY SELECTION PROCESS

During the initial search, we found 196 different papers, although 11 of them were duplicates that we immediately disqualified. All original papers were reviewed using the principles outlined in Table5 and information available on the first page of the paper. In this manner, we removed all works not connected to the research field, collected from unreliable sources, or with other weaknesses. Examination of this kind identified 47 entries that did not meet the inclusion criteria

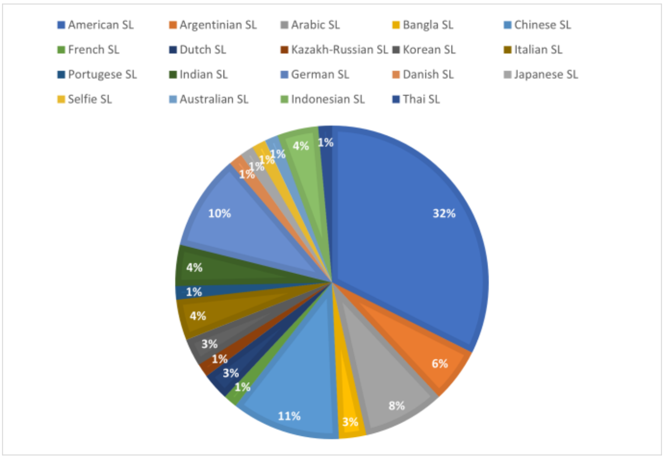


Figure 6.2 :Number of publication on Sign Language Recognition by Language.

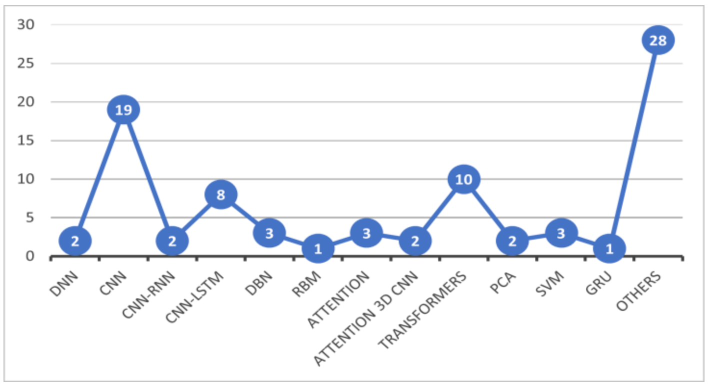


Figure 6.3: Number of publications on Sign Language Recognition by Architecture 138 core and relevant studies remained.

We examined the full text of the studies next, and removed any that failed to directly address SLR or to support their hypothesis with high-quality data removed as well. Next, we took the quality of all quotes and correct naming of sources into account, and performed online checks to ascertain the authorship of source papers. In the last phase, we performed a qualitative evaluation to determine which studies deserved to be reported on. The entire selection procedure cut down the number of included works to 84, but their level of scientific value and importance regarding the main research questions leaves nothing to be desired.

6.4 SLR MODEL TYPES

There are two types of models related to the recognition process of sign languages: the isolated model and the continuous model. In the following sections we will show the work that has been done in this aspect.

6.4.1 SLR CONTINUOUS MODELS

As part of sign language recognition and modeling, some experiments have used continuous models. For example, Wu et al. [38] proposed a new bimodal dynamic network suitable for continuous recognition of gestures. The model relied on the positions of the 3D joints, as well as audio utterances of the gesture tokens. Koller et al. [63] demon- strated the use of an EM-based algorithm for continuous sign language recognition. The EM-based algorithm was designed to address the temporal alignment problem associated with continuous video processing tasks. Similarly, Li et al. [42] proposed a framework that addresses some of the scalability challenges associated with continuous sign language recog- nition. Another experiment by Camgoz et al. [64] developed an end-to-end system designed for continuous sign language alignment and recognition. The model is based on explicit subunit modeling. Similarly, Wang et al. [66] suggested a connectionist temporal fusion method having the capability to translate continuous visual languages in videos into textual language sentences.

Additional studies on continuous SLR models have been conducted by Rao et al. [68]. A system was developed and evaluated at various times using continuous Indian Sign Language sentences developed from 282 words. Similarly, Koller et al. [77] used a database consisting of continuous signing in German Sign Language. In [46], animations were processed continuously. However, this approach proved to be extremely challenging because the animations were difficult to work with after processing. While exploring the challenges of continuous translation, Pigou et al. [126] observed that deep residual networks can be used to learn patterns in con- tinuous videos containing gestures and signs. The use of deep residual networks can minimize the need for preprocessing. In [17], a model was developed that can enhance existing sign language recognition methods by between 15% and 38% relatively, and by 13.3% absolutely. Cui et al. [133] also suggested a weakly supervised approach that could recognize sign language continuously with the help of deep neural networks. This approach achieved an outcome comparable to state-of-the-art approaches

6.4.2 SLR ISOLATED MODELS

Until recently, a majority of sign language recognition exper iments have been carried out on isolated sign samples. These models examine a sequence of images or signals based on hand movements obtained from sensor gloves [97]. Sensor gloves often represent a complete sign. For instance, Koller et al. [63] used a dataset that featured isolated signs from Dan- ish and New Zealand sign languages. Another experiment by [37] proposed an isolated SLR system designed to extract .discriminative aspects from videos, where each signed video corresponded to one word. After evaluating the challenges of continuous translation, Escudeiro et al. [46] resorted to an isolated approach. In essence, every gesture was created separately, making it easier to use animations with ease. Different observations by Fang et al. [128] suggested the use of a hierarchical model reliant on deep recurrent neural networks. The model successfully combined the isolated lowlevel American Sign Language characteristics into an organized high-level representation that could be used for translation.

Recent developments in sign language experiments have also suggested that the use of regions of interest (ROIs) to isolate hand gestures and sign language features can enhance the accuracy of recognition [134]. In [131], the authors used an isolated gloss recognition system to facilitate real- time sign language translation. The isolated gloss recognition system included video pre-processing as well as a time- series neural network module. Another experiment by Latif et al. [169] also considered video segments based on an es- timated “gloss-level.” While making their observations, Cui et al. [3] set their receptive field to the estimated length of an isolated sign. A recent study by Huang et al. [49] focused on a basic isolated sign language recognition task. The use of an attention-based 3D-CNN was proposed to recognize a large vocabulary. The model was advantageous because of it took advantage of the spatio-temporal feature learning capabilities of the 3D-CNN. Papadimitriou et al. [95] used the American Sign Language Lexicon Video Dataset, which consists of video sequences of isolated American Sign Language signs.

6.4.3 DELIBERATIONS ABOUT CONTINUOUS AND ISOLATED SLR

SLR comprises two distinct modes – isolated and continuous, each of which requires a different approaches and is associ- ated with very specific challenges. In particular, one key dis- tinction is that direct supervision is much more essential for continuous SLR. In isolated SLR, all the relevant content is concentrated in a limited area of a single image, but in contin- uous SLR it is necessary to carefully align the sections of the video in chronological order and ensure that each sentence is properly tagged. That’s one example of the complexities associated with continuous sign language recognition, which is far more demanding in terms of computing efficiency. This must be taken into account during the evaluation of methodologies, as well as the feature selection process. If sequential labeling is done correctly and the most predictive features are selected, the resulting model has a higher chance of being accurate with continuous video analysis.

Over the last several years, smart applications of deep learning systems have removed many obstacles in this field as well as many other related automation tasks, but real breakthroughs that could lead to broad application by general population are still ahead. The attention mechanism is an intriguing element that works well with different types of data, and can be used to describe complex interactions in space and time (for example, Graph Neural Networks appli- cations). Further research will show whether this approach is the most optimal for resolving the issues complicating continuous SLR at the moment.

6.4.4 SIGN LANGUAGE RECOGNITION BASED ON LOCALIZATION

Many basic concepts surround the use of sign language. First,sign languages are never international. Most, but not all, nations use different sign languages. Sign language is popular in American, British, Arabic, and Chinese settings, among many others. Table 3 provides an overview of various stud- ies undertaken using different sign languages. For instance, American Sign Language (ASL), the most popular localiza- tion, includes independent grammar rules that are not a visual form of English. Application of this localization was evident in the experiment by Rioux-Maldague et al. [44], where the authors applied their proposed technique and classified ASL based on grammatical rules. Another experiment by Tang et al. [39] considered 36 hand postures obtained from American Sign Language to facilitate posture training and recognition. However, there are other systems that derive non-ASL signs and use them in English order. Such examples include exper- iments that have focused on Italian Sign Language. In [38], a dataset consisting of 20 Italian cultural or anthropological signs was used to evaluate a novel bimodal dynamic network designed to recognize gestures. The Italian dataset consisted of 393 labeled sequences and a total of 7,754 gestures.

Arabic Sign Language is also considered the preferred communication approach among many deaf people. In [40], depth and intensity images in the Arabic language were used to develop a system that can recognize associated signs . The proposed system was tested using a dataset obtained from thre different users, resulting in an accuracy of 99.5%. The authors in [121], [169] also used an Arabic Sign Lan- guage dataset. In some cases, sign language experiments have focused on Chinese. In [81], vocabulary was adopted from 510 distinct words obtained from Chinese Sign Lan- guage. Among these words, 353 were single-sign words, while the remaining were multi-sign words. Yang et al. [65] also showed interest in Chinese Sign Language and used the instructional video We Learn Sign Language to meet the objective of their experiment. Another experiment by Jiang et al. [71] used Chinese Sign Language to facilitate the fingerspelling process. Furthermore, the authors in [49]–[51],[97] used Chinese Sign Language in their experiments. A few other experiments evaluated Argentine Sign Lan- guage. An example would be [37], where an initial dataset was obtained from 10 subjects speaking Argentine Sign Language. Konstantinidis et al. [70] also used Argentine Sign Language featuring 10 subjects to explore hand and body skeletal recognition. Rather than focusing on a single language, some experiments use a mixture of sign languages. For example, Koller et al. [63] employed a mixture of Danish and New Zealand sign languages in their effort to examine how to train a CNN on 1 million hand images. The sign languages were obtained from two representative videos based on publicly available lexicons. The Danish data did not have any motion blur, while the New Zealand version had some motion blur. In another experiment, Camgoz et al. [64] focused on Danish, New Zealand, and German sign languages to evaluate the role of SubUNets in sign language recognition.

CHAPTER 7
UML DIAGRAM

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

7.1 USE CASE DIAGRAMS

A use case diagram is used to represent the dynamic behaviour of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system.

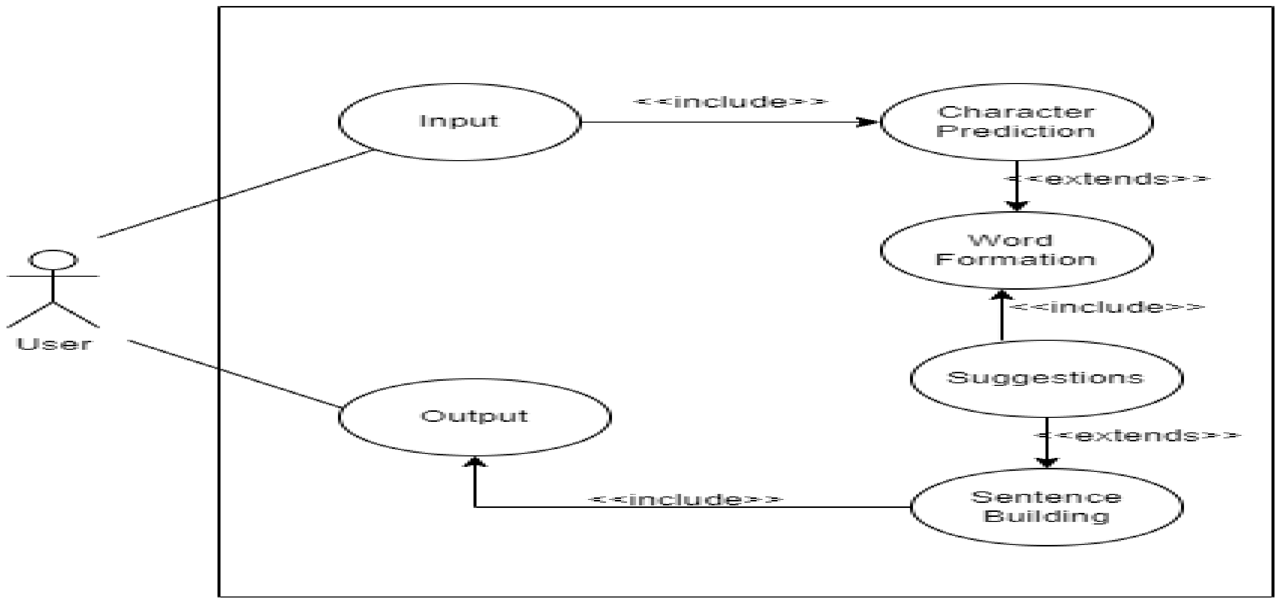


Figure 7.1: USE Case Diagram

7.2 DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both. It shows how data enters and leaves the system, what changes the information, and where data is stored.

7.2.1 LEVEL 1 - DFD

In 1-level DFD, a context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main objectives of the system and breakdown the high-level process of 0-level DFD into subprocesses.

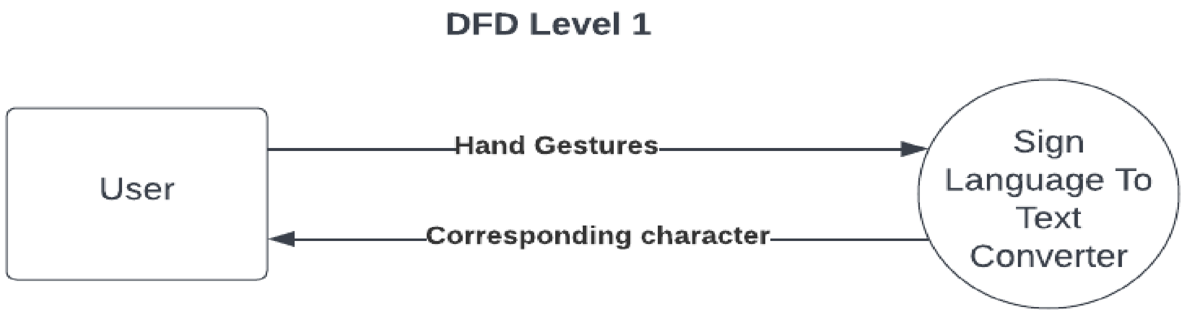


Figure 7.2:Level-1 DFD

7.2.2 LEVEL 2 – DFD

In 2-level DFD goes one process deeper into parts of 1-level DFD. It can be used to project or record the specific/necessary detail about the system's functioning. This level provides an even more detailed view of the system by breaking down the sub-processes identified in the level 1 DFD into further sub-processes. Each sub-process is depicted as a separate process on the level 2 DFD. The data flows and data stores associated with each sub-process are also shown.

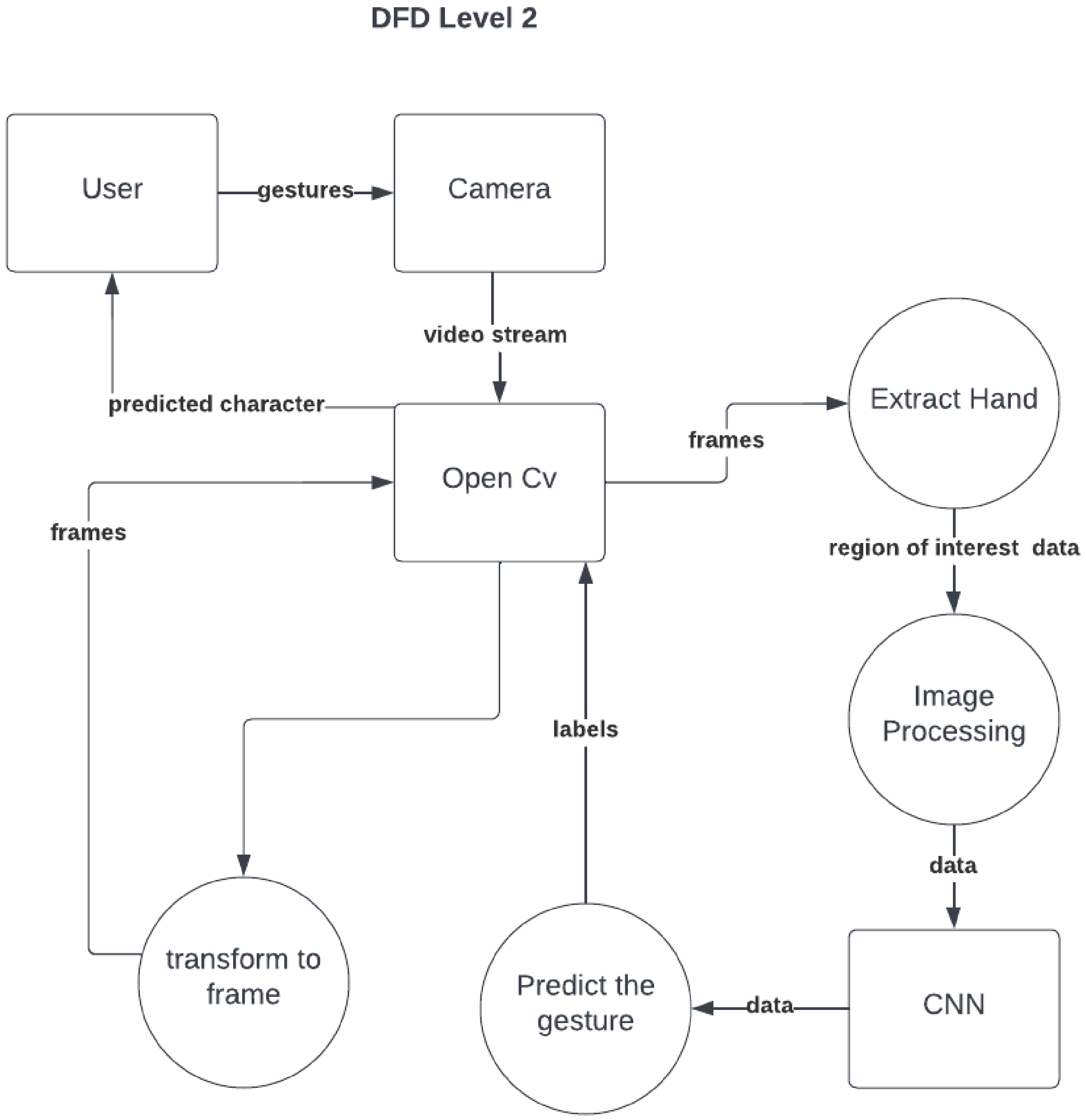


Figure 7.3: Level – 2 DFD

7.3 SEQUENCE DIAGRAM

A sequence diagram is a type of interaction diagram because it describes how—and in what order a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process.

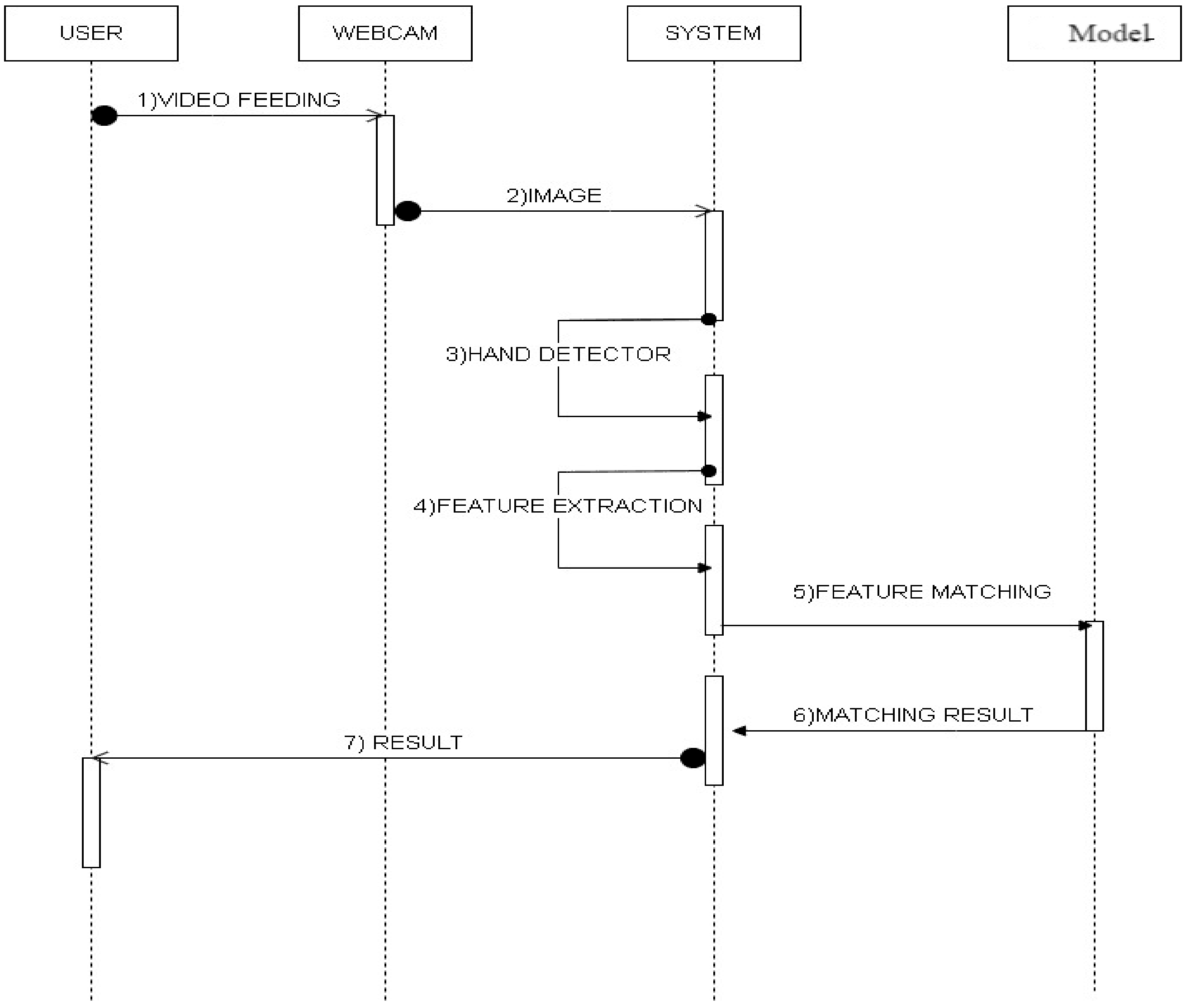


Figure 7.4:Sequence Diagram

CHAPTER 8

ALGORITHMS

Classification machine learning algorithms like SVM, k-NN are used for supervised learning, which involves labelling the dataset before feeding it into the algorithm for training. For this project, various classification algorithms are used: CNN. Feature extraction algorithms are used for dimensionality reduction to create a subset of the initial features such that only important data is passed to the algorithm. When the input to the algorithm is too large to be processed and is suspected to be redundant (like repetitiveness of images presented by pixels), then it can be converted into a reduced set of features. Feature extraction algorithms: PCA, LBP, and HoG, are used alongside classification algorithms for this purpose. This reduces the memory required and increases the efficiency of the model.

8.1 CNN

Convolutional Neural Networks (CNN), are deep neural networks used to process data that have a grid-like topology, e.g., images that can be represented as a 2-D array of pixels. A CNN model consists of four main operations: Convolution, Non-Linearity (Relu), Pooling and Classification (Fully-connected layer).

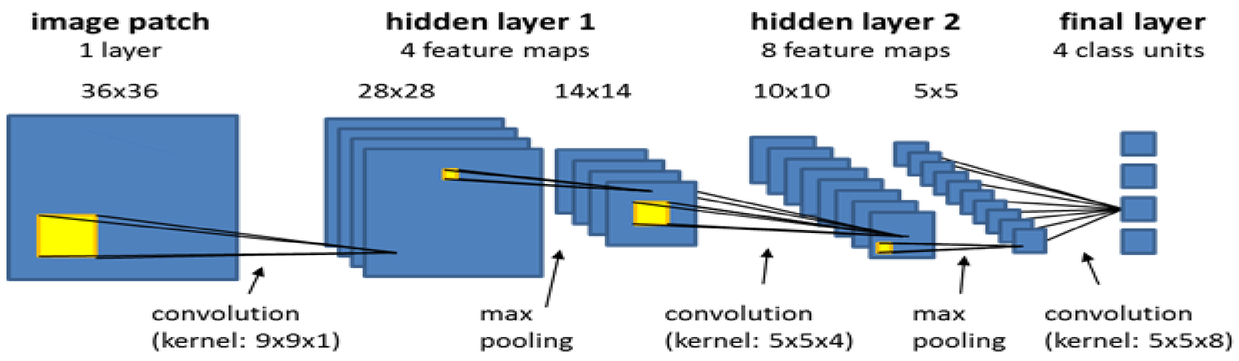


Figure 8.1 :CNN

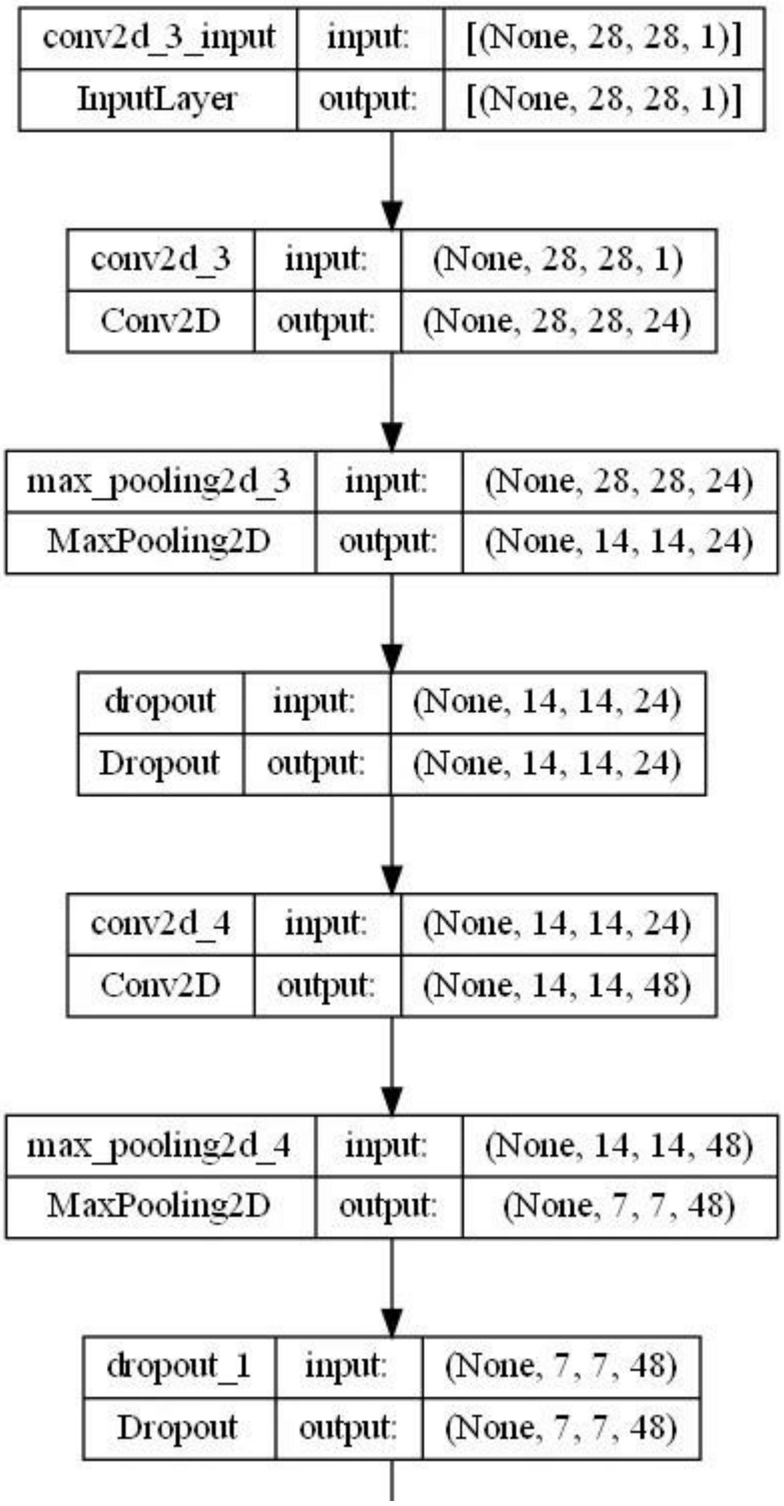

conv2d_3_input | input: | [(None, 28, 28, 1)]
InputLayer | output: | [(None, 28, 28, 1)]
conv2d_3 | input: | (None, 28, 28, 1)
Conv2D | output: | (None, 28, 28, 24)
max_pooling2d_3 | input: | (None, 28, 28, 24)
MaxPooling2D | output: | (None, 14, 14, 24)
dropout | input: | (None, 14, 14, 24)
Dropout | output: | (None, 14, 14, 24)
conv2d_4 | input: | (None, 14, 14, 24)
Conv2D | output: | (None, 14, 14, 48)
max_pooling2d_4 | input: | (None, 14, 14, 48)
MaxPooling2D | output: | (None, 7, 7, 48)
dropout_1 | input: | (None, 7, 7, 48)
Dropout | output: | (None, 7, 7, 48)

Figure 8.2 : Architecture Of Model using CNN

The diagram appears to represent the architecture of a Convolutional Neural Network (CNN) as part of a deep learning model, typically used for image processing tasks.

Here’s a breakdown of each layer as depicted in the diagram:

InputLayer:

This layer, often the starting point of a CNN, takes the input image. The shape ([None, 28, 28, 1]) indicates it expects an image of size 28x28 pixels with a single color channel (grayscale).

Conv2D (conv2d_3):

This is a convolutional layer that applies 24 filters to the input, capturing various features like edges, textures, etc. The output shape ([None, 28, 28, 24]) reflects that there are now 24 feature maps, one for each filter.

MaxPooling2D (max_pooling2d_3):

This pooling layer reduces the spatial dimensions of the input feature maps by half, resulting in a size of 14x14 pixels. It takes the maximum value over a 2x2 pixel window and moves over the feature maps, hence reducing the dimensionality and retaining the most significant features.

Dropout (dropout):

A dropout layer randomly sets a fraction of the input units to 0 at each update during training time, which helps prevent overfitting. The shape remains unchanged (([None, 14, 14, 24])), but some of the nodes are "dropped" or deactivated.

Conv2D (conv2d_4):

Another convolutional layer that increases the depth of the network by applying 48 filters. This further processes the data to extract more complex features, resulting in an output shape of ([None, 14, 14, 48]).

MaxPooling2D (max_pooling2d_4):

Similar to the earlier max pooling, it further reduces the dimensionality of the feature maps, this time down to a 7x7 grid, still with 48 depth due to the filters.

Dropout (dropout_1):

Another dropout layer is applied to further mitigate overfitting, keeping the shape ([None, 7, 7, 48]) intact but deactivating a fraction of the nodes.

Each layer operates on the output of the previous layer, progressively transforming the input image into a form that is easier to classify. The convolutional layers are responsible for feature extraction, while the pooling layers reduce the spatial size of the representation to decrease the number of parameters and computation in the network. Dropout layers are used to prevent overfitting, ensuring that the network remains robust and can generalize well when it sees new, unseen images.

8.2 PCA (Principal Component Analysis)

Using PCA, data is projected to a lower dimension for dimensionality reduction. The most important feature is the one with the largest variance or spread, as it corresponds to the largest entropy and thus encodes the most information. Thus, the dimension with the largest variance is kept while others are reduced.

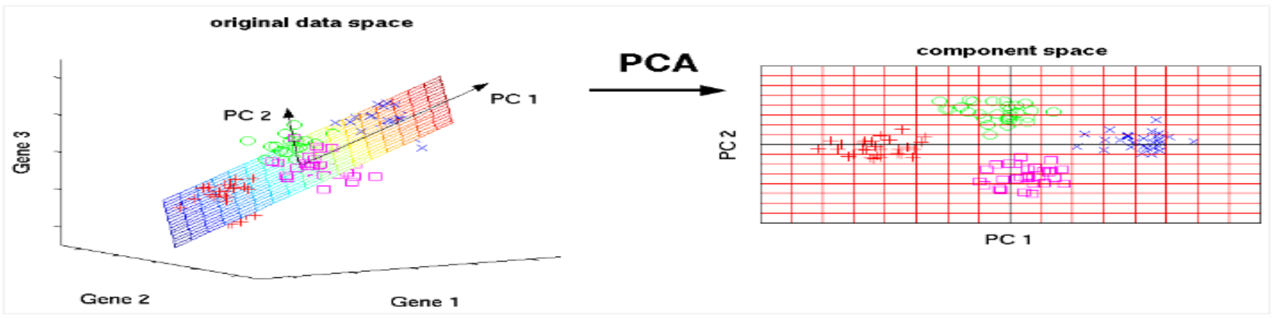


Figure 8.3:PCA feature reduction

Principal Component Analysis (PCA) is a statistical technique used for dimensionality reduction while preserving as much variability as possible. It transforms a set of possibly correlated variables into a smaller number of uncorrelated variables called principal components. The first principal component captures the greatest variance in the data, followed by the second, and so on, each orthogonal to the others. PCA begins by standardizing the data, then constructs a covariance matrix to analyze the correlation between features. From this matrix, eigenvectors and eigenvalues are derived, with eigenvectors representing the directions of maximum variance and eigenvalues indicating the magnitude of these directions. By sorting these eigenvalues in descending order and selecting the top components, PCA reduces the dimensionality of the data, projecting it onto a new space with reduced dimensions but maintaining the essence of the original data. This method is particularly useful in feature extraction, noise reduction, and data visualization. It’s widely used as a preprocessing step in machine learning to enhance model performance by removing multicollinearity and reducing overfitting. However, PCA assumes linearity and that high variance directions are more informative, which may not always hold true, and it can be sensitive to outliers in the data.

8.3 LBP (Local Binary Patterns)

LBP computes a local representation of texture which is constructed by comparing each pixel by its surrounding or neighbouring pixels. The results of this are stored as an array which is then converted into decimal and stored as an LBP 2D array.

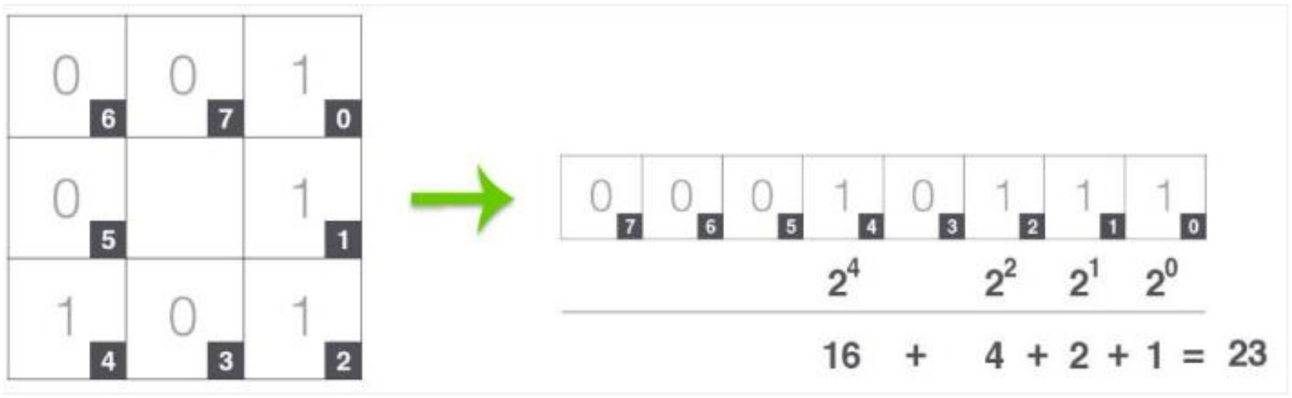
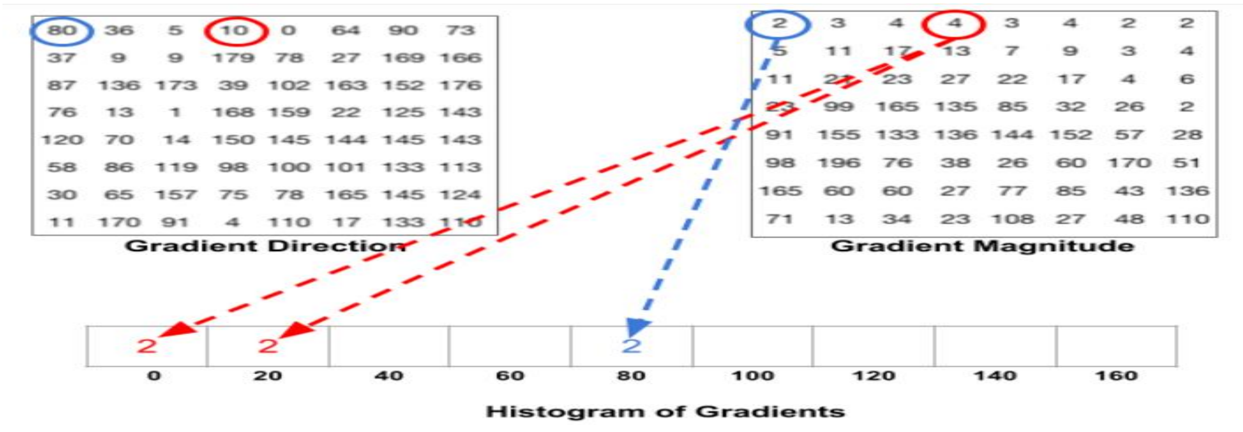


Figure 8.4 :Conversion of pixel into LBP representation

It is a texture descriptor widely used in computer vision for analyzing the local texture features of images. It is particularly effective in applications such as facial recognition, motion analysis, and medical image analysis. The LBP algorithm works by selecting a central pixel and comparing its intensity with that of its neighbors. If a neighbor's intensity is higher or equal, it is marked as 1; otherwise, it is marked as 0. This comparison results in a binary number for each pixel, capturing the texture around it. The histogram of these binary numbers across the image serves as a robust descriptor of its texture. LBP is highly valued for its resistance to changes in lighting and its computational simplicity, making it suitable for real-time processing. Despite its strengths, LBP can be sensitive to noise, and it primarily captures texture information, possibly requiring supplementation with other methods for color-sensitive tasks. Its applications span from enhancing facial recognition systems by identifying features like eyes and mouths to analyzing tissue structures in medical imaging, proving its versatility and effectiveness in various image processing contexts.

8.4 HoG (Histogram of Gradients)

Hog is a feature descriptor that calculates a histogram of gradient for the image pixels, which is a vector of 9 bins (numbers) corresponding to the angles: 0, 20, 40, 60... 160. The images are divided into cells, (usually, 8x8), and for each cell, gradient magnitude and gradient angle is calculated, using which a histogram is created for a cell. The histogram of a block of cells is normalized, and the final feature vector for the entire image is calculated.



Gradient Direction

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Magnitude

2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Histogram of Gradients

0	20	40	60	80	100	120	140	160
2	2			2				

Figure 8.5 : Creating histogram from Gradient of magnitude and direction

It is a feature descriptor extensively used in the field of computer vision for object detection by capturing gradient orientation information in localized sections of an image. This process involves first computing the gradients of the image to determine the direction and strength of edges by applying filters such as the Sobel operator. The image is then divided into small spatial regions called cells, where a histogram of gradient directions is compiled for each cell. These histograms plot the distribution of edge directions within the cell, aggregating the gradient magnitudes that fall into each orientation bin. To enhance robustness against variations in illumination and contrast, the histograms are normalized over larger, overlapping blocks of cells. This normalization helps to reduce the effects of lighting differences across the image. The final feature descriptor is formed by concatenating these normalized histograms from all the blocks, producing a comprehensive representation of the image's local shape and texture features. HoG descriptors have been pivotal in applications such as pedestrian detection, where their ability to outline human forms by capturing edge and silhouette information is particularly valuable. However, while HoG offers robustness to changes in geometry and lighting, its performance can degrade under object rotation and it is computationally demanding, which has led to it being supplemented or replaced by deep learning techniques like Convolutional Neural Networks in more complex or large-scale vision tasks.

CHAPTER 9
METHODOLOGY

Ultramodern technology is sophisticated and able of helping the deaf and hard of hail. We propose a model that can comprehend sign language and uses the Convolutional network The predicted words will be shown in real- time when sign language is detected using a smartphone camera. The model can also be made available for a variety of sign languages.

9.1 DATA ACQUISITION

Gather a diverse and comprehensive dataset of sign language gestures. This dataset should include various hand symbols, signing styles, and movements performed by individuals with different backgrounds. It is crucial to ensure that the dataset represents the target sign language(s) accurately. The following are some techniques for learning about hand motions using various methods: To enable precise hand configuration and position, it employs electromechanical devices. Information extraction techniques based on gloves might vary. But it is expensive and not user-friendly. The input device for seeing the information of hands and/or fingers in vision-based approaches is the computer webcam. In order to realize natural contact between people and computers without the need for any additional equipment, the Vision Based approaches simply need a camera, which lowers expenses. The main difficulty in vision-based hand detection is dealing with the enormous variability in the appearance of the human hand caused by a great number of hand movements, the potential for different skin tones, as well as the various viewpoints, scales, and camera shutter speeds used to capture the scene. Many active and intrusive cameras are utilized to capture the depth data while one camera is used to collect standard signs.

9.2 DATA PRE-PROCESSING AND FEATURE EXTRACTION

In this approach for hand detection, firstly we detect hand from image that is acquired by webcam and for detecting a hand we used media pipe library which is used for image processing. So, after finding the hand from image we get the region of interest (Roi) then we cropped that image and convert the image to Gray image using OpenCV library after we applied the gaussian blur. The filter can be easily applied using open computer vision library also known as OpenCV. Then we converted the gray image to binary image using threshold and Adaptive threshold methods. In this method there are many loop holes like your hand must be ahead of clean soft background and that is in proper lightning condition then only this method will give good accurate results but in real world we don't get good background everywhere and we don't get good lightning conditions too. So, to overcome this situation we try different approaches then we reached at one interesting solution in which firstly we detect hand from frame using mediapipe and get the hand landmarks of hand present in that image then we draw and connect those landmarks in simple white image.

Mediapipe Landmark System:

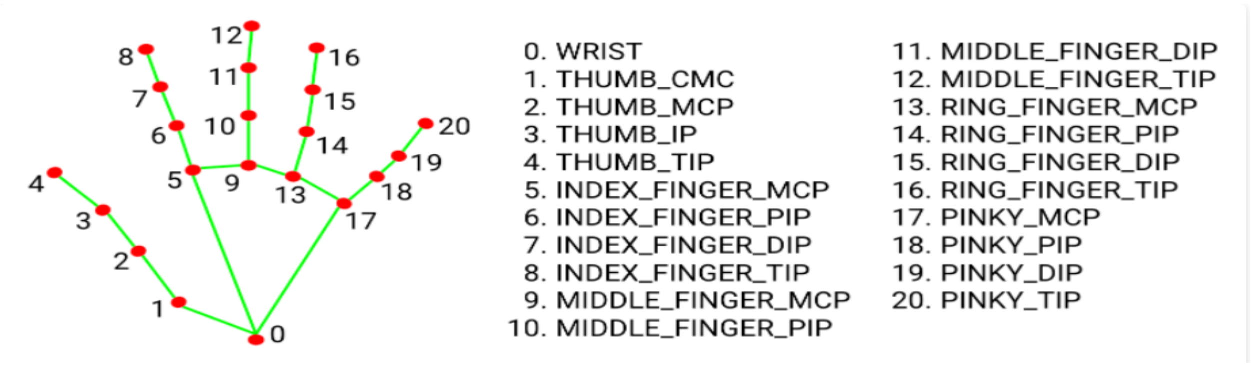


Figure 9.1: Landmark System

Now we get these landmark points and draw it in plain white background using opencv library. By doing this we tackle the situation of background and lightning conditions because the mediapipe library will give us landmark points in any background and mostly in any lightning conditions.

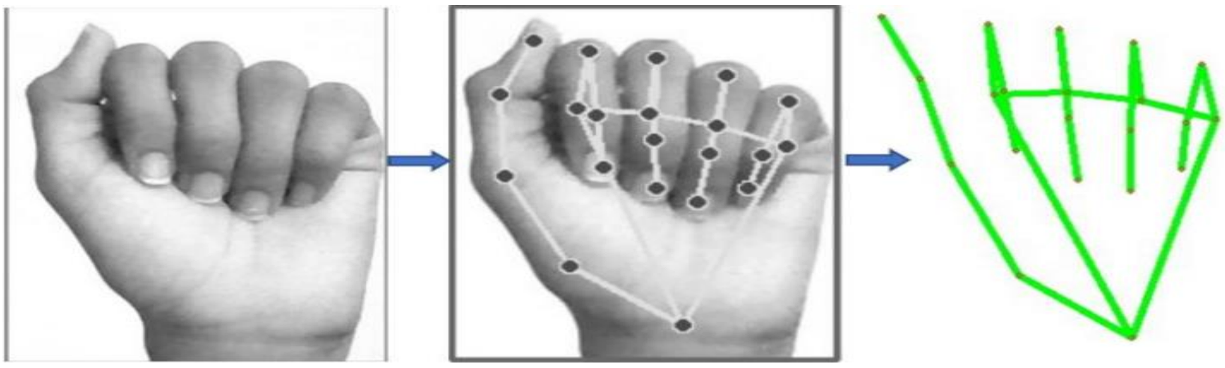


Figure 9.2 : Landmark Processing system using OpenCV We have collected 520 skeleton images of wods

9.3 GESTURE CLASSIFICATION

9.3.1 Convolutional Neural Network (CNN)

Convolutional Layer - In convolution layer I have taken a small window size [typically of length 5*5] that extends to the depth of the input matrix. The layer consists of learnable filters of window size. During every iteration I slid the window by stride size [typically 1], and compute the dot product of filter entries and input values at a given position. As I continue this process well create a 2-Dimensional activation matrix that gives the response of that matrix at every spatial position. That is, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some colour.

9.3.2 - We use pooling layer to decrease the size of activation matrix and ultimately reduce the learnable parameters.

There are two types of pooling:

(a) Max Pooling - In max pooling we take a window size [for example window of size 2*2], and only taken the maximum of 4 values. Well lid this window and continue this process, so well finally get an activation matrix half of its original Size.

(b) Average Pooling: To construct a down sampled (pooled) feature map, the average value for patches of a feature map is calculated using the average pooling method. In average pooling we take average of all Values in a window.

9.3.3 Fully Connected Layer - In convolution layer neurons are connected only to a local region, while in a fully connected region, well connect the all the inputs to neurons. The pre-processed 180 images/words will feed the karas CNN model. Because we got bad accuracy in 26 different classes thus, we divided whole words into 8 classes in which every class contains similar words. All the gesture labels will be assigned with a probability. The label with the highest probability will be treated to be the predicted label. So, when model will classify [aemnst] in one single class using mathematical operation on hand landmarks we will classify further into single words. Finally, we got 97% Accuracy (with and without clean background and proper lightning conditions) through our method. And if the background is clear and there is good lightning condition then we got even 99% accurate results.

9.4 TEXT TO SPEECH TRANSLATION

The model translates known gestures into words. we have used pyttsx3 library to convert the recognized words into the appropriate speech. The text-to-speech output is a simple workaround, but it's a useful feature because it simulates a real-life dialogue.

The entry field text value, which represents the phrases or text to be read, will be stored in the message variable. To read the text, lang uses that language. English is used as the default language.

CHAPTER 10
IMPLEMENTATION AND RESULT

10.1 DATA SET

We created our dataset (IISL2020) for training and testing. The dataset was collected from 16 subjects aged between 20 and 25 and included both female and male participants. We obtained the relevant data for specific dynamic-based isolated gestures. All the video samples have an average of 28 FPS and a resolution of 1920 1080, with an average length of 2 s. The dataset consisted of 11 words; for each word, about 1100 video samples were created with the 16 subjects. This dataset was created under natural conditions—without any extra brightness, orientation,. Furthermore, the authors did not use any external help from sensors or smart gloves to detect hand movements, as shown in

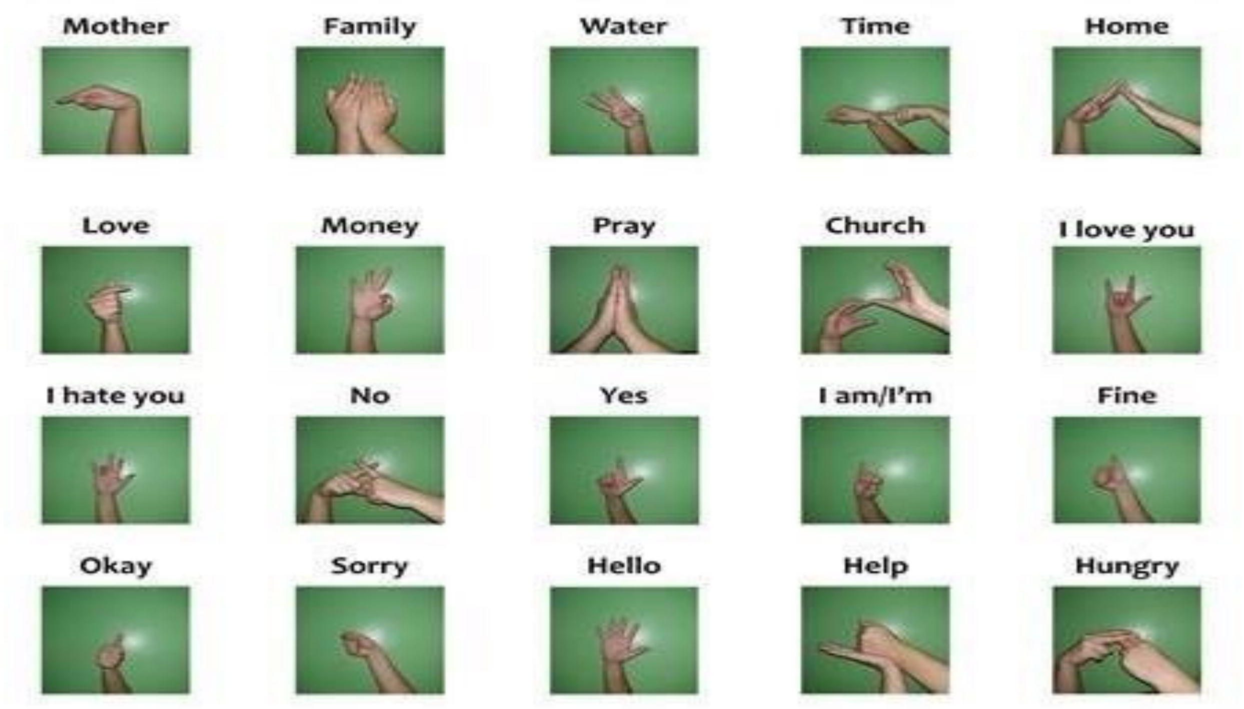


Figure10.1SLR of Words

10.2 RESULT AND DISCUSSION

Each curve's cross-validation score is poor and becomes worse with time. When comparing the random forest's performance to that of the decision tree and the naive Bayes classifier, the curve indicates very good performance. The training score is almost at its maximum in decision trees and random forests, but it gradually declines in the case of naive Bayes. Figure 10 displays the learning curves for the ASL-words. It is evident that, when compared to the other four classifiers, the random forest classifier performs better using the suggested approach.

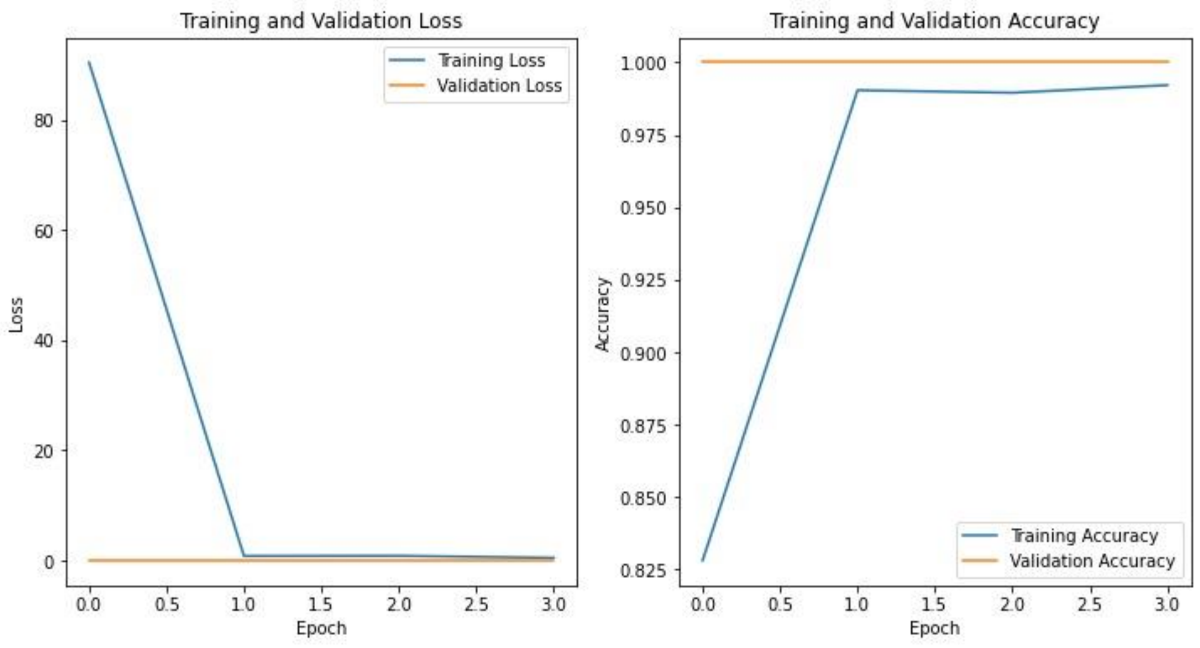
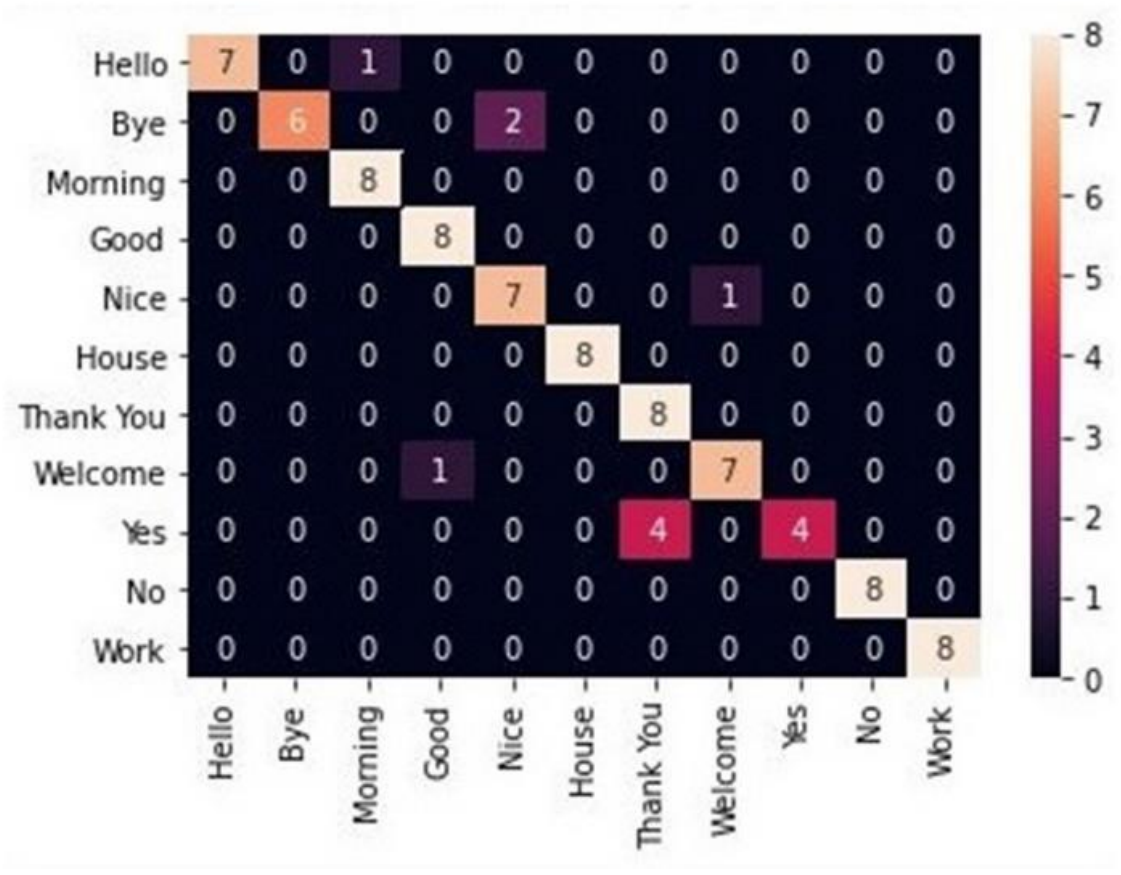


Figure 10.2: Accuracy score on the y-axis and training examples on the x-axis

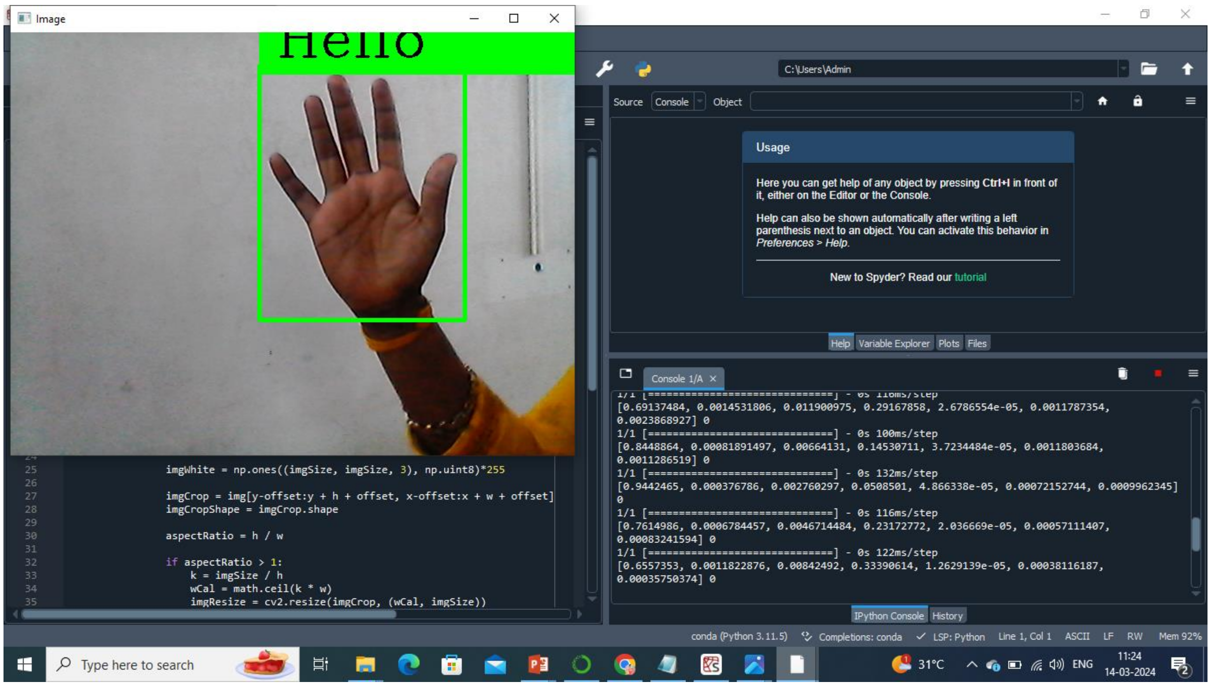
A confusion matrix for American Sign Language (ASL) recognition represents the performance of a classification model by showing the predicted labels against the true labels for each class (ASL letter gesture). It provides detailed information about the accuracy and misclassifications of the system. Moreover, Naive Bayes, which performs best with independent features, has the lowest accuracy rates. Yet, the traits are interdependent in the suggested strategy. The angle between landmark 12 and landmark 4, for instance, changes when the angle between landmark 0 and landmark 12 (12 is located on the tip of the middle finger), which influences the other characteristics as well. From the confusion matrix, we can calculate various performance metrics such as accuracy, precision, recall, and F1score. It provides insights into which ASL gestures are more challenging for the system to recognize and helps identify potential areas for improvement in the recognition model or training process. The accuracy of the chosen ASL words datasets, as shown by the confusion matrices:



	Hello	Bye	Morning	Good	Nice	House	Thank You	Welcome	Yes	No	Work
Hello	7	0	1	0	0	0	0	0	0	0	0
Bye	0	6	0	0	2	0	0	0	0	0	0
Morning	0	0	8	0	0	0	0	0	0	0	0
Good	0	0	0	8	0	0	0	0	0	0	0
Nice	0	0	0	0	7	0	0	1	0	0	0
House	0	0	0	0	0	8	0	0	0	0	0
Thank You	0	0	0	0	0	0	8	0	0	0	0
Welcome	0	0	0	1	0	0	0	7	0	0	0
Yes	0	0	0	0	0	0	4	0	4	0	0
No	0	0	0	0	0	0	0	0	0	8	0
Work	0	0	0	0	0	0	0	0	0	0	8

Figure 10.3 : Confusion Matrix of 27 one-hand ASL words.

10.3 OUTPUT



Usage

Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

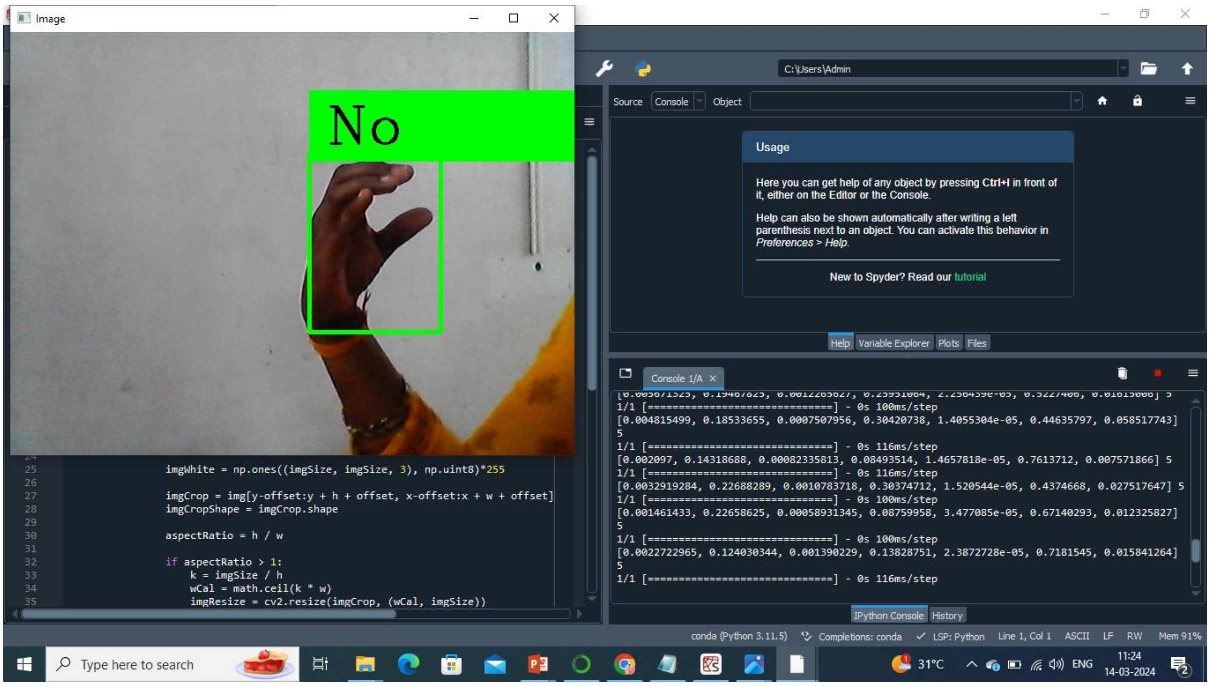
New to Spyder? Read our tutorial

```
            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))
```

```
[0.69137484, 0.0014531806, 0.011900975, 0.29167858, 2.6786554e-05, 0.0011787354,
0.0023868927] 0
1/1 [==============================] - 0s 100ms/step
[0.8448864, 0.00081891497, 0.00664131, 0.14530711, 3.7234484e-05, 0.0011803684,
0.0011286519] 0
1/1 [==============================] - 0s 132ms/step
[0.9442465, 0.000376786, 0.002760297, 0.0508501, 4.866338e-05, 0.00072152744, 0.0009962345]
0
1/1 [==============================] - 0s 116ms/step
[0.7614986, 0.0006784457, 0.0046714484, 0.23172772, 2.036669e-05, 0.00057111407,
0.00083241594] 0
1/1 [==============================] - 0s 122ms/step
[0.6557353, 0.0011822876, 0.00842492, 0.33390614, 1.2629139e-05, 0.00038116187,
0.00035750374] 0
```

conda (Python 3.11.5) Completions: conda LSP: Python Line 1, Col 1 ASCII LF RW Mem 92%

10.4: Output For Hello Sign



Usage

Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

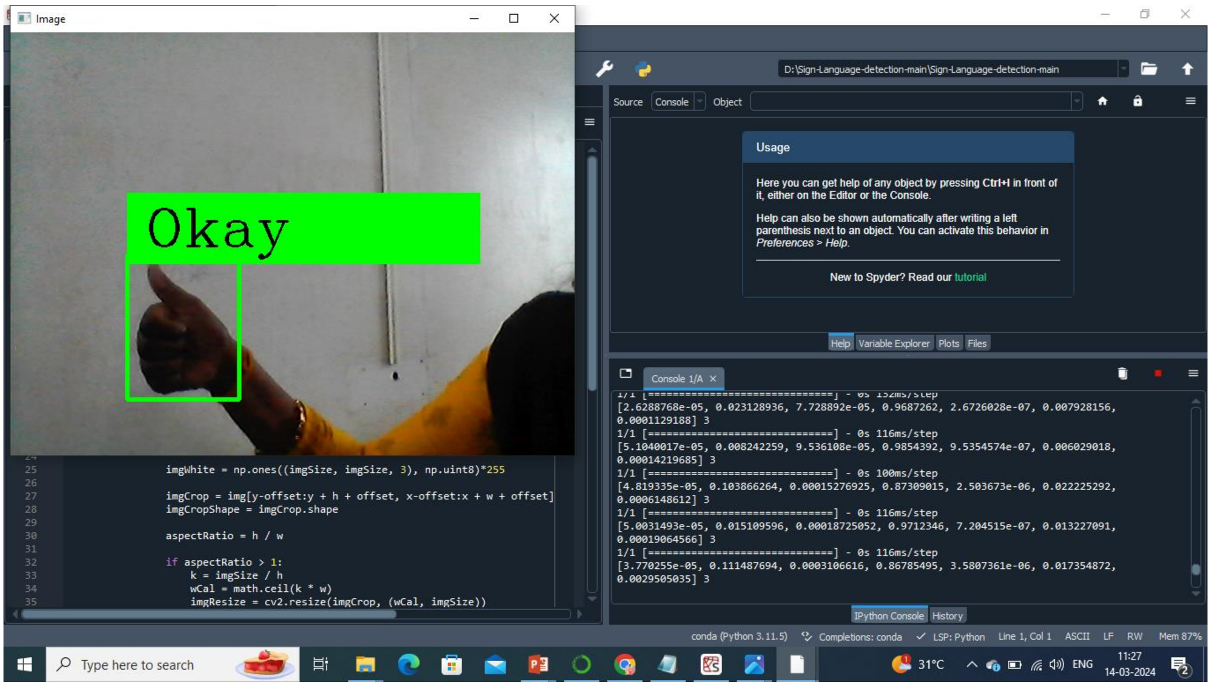
New to Spyder? Read our tutorial

```
            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))
```

```
1/1 [==============================] - 0s 100ms/step
[0.004815499, 0.18533655, 0.0007507956, 0.30420738, 1.4055304e-05, 0.44635797, 0.058517743]
5
1/1 [==============================] - 0s 116ms/step
[0.002097, 0.14318688, 0.00082335813, 0.08493514, 1.4657818e-05, 0.7613712, 0.007571866] 5
1/1 [==============================] - 0s 116ms/step
[0.0032919284, 0.22688289, 0.0010783718, 0.30374712, 1.520544e-05, 0.4374668, 0.027517647] 5
1/1 [==============================] - 0s 100ms/step
[0.001461433, 0.22658625, 0.00058931345, 0.08759958, 3.477085e-05, 0.67140293, 0.012325827]
5
1/1 [==============================] - 0s 100ms/step
[0.0022722965, 0.124030344, 0.001390229, 0.13828751, 2.3872728e-05, 0.7181545, 0.015841264]
5
1/1 [==============================] - 0s 116ms/step
```

conda (Python 3.11.5) Completions: conda LSP: Python Line 1, Col 1 ASCII LF RW Mem 91%

10.5:Output For No Sign



Usage

Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.

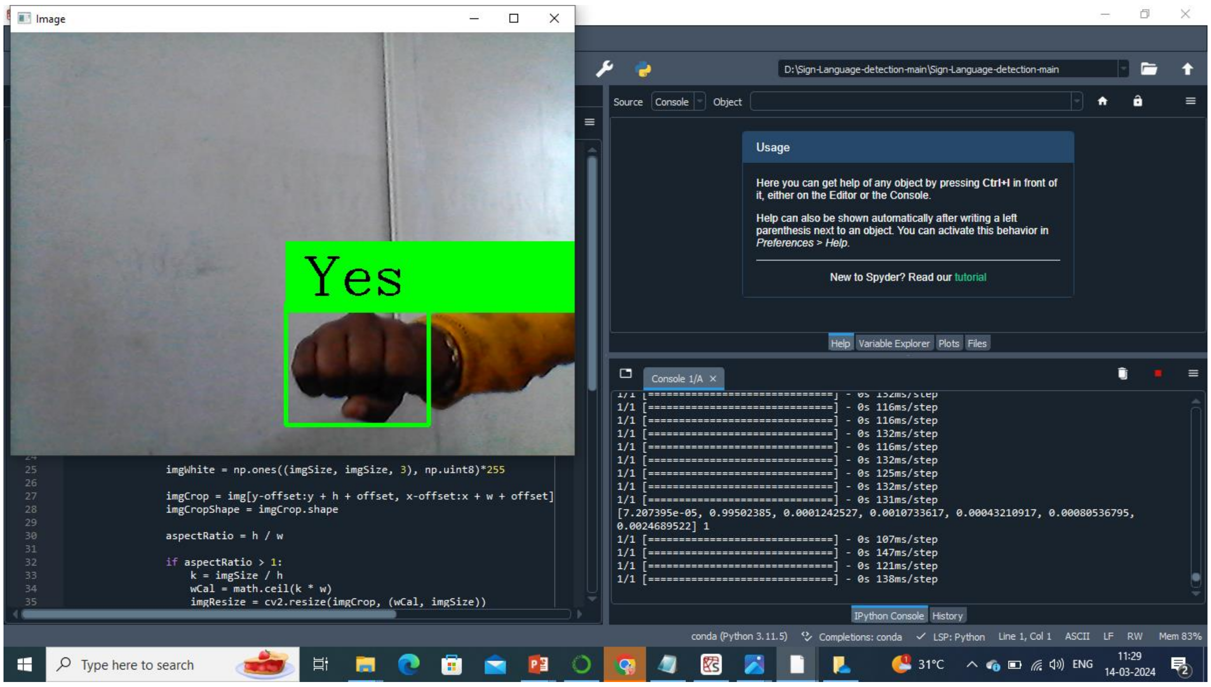
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
            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))
```

```
1/1 [==============================] - 0s 132ms/step
[2.6288768e-05, 0.023128936, 7.728892e-05, 0.9687262, 2.6726028e-07, 0.007928156,
0.0001129188] 3
1/1 [==============================] - 0s 116ms/step
[5.1040017e-05, 0.008242259, 9.536108e-05, 0.9854392, 9.5354574e-07, 0.006029018,
0.00014219685] 3
1/1 [==============================] - 0s 100ms/step
[4.819335e-05, 0.103866264, 0.00015276925, 0.87309015, 2.503673e-06, 0.022225292,
0.0006148612] 3
1/1 [==============================] - 0s 116ms/step
[5.0031493e-05, 0.015109596, 0.00018725052, 0.9712346, 7.204515e-07, 0.013227091,
0.00019064566] 3
1/1 [==============================] - 0s 116ms/step
[3.770255e-05, 0.111487694, 0.0003106616, 0.86785495, 3.5807361e-06, 0.017354872,
0.0029505035] 3
```

10.6:Output For Okay Sign



Usage

Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
            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))
```

```
1/1 [==============================] - 0s 132ms/step
1/1 [==============================] - 0s 116ms/step
1/1 [==============================] - 0s 116ms/step
1/1 [==============================] - 0s 132ms/step
1/1 [==============================] - 0s 116ms/step
1/1 [==============================] - 0s 132ms/step
1/1 [==============================] - 0s 125ms/step
1/1 [==============================] - 0s 132ms/step
1/1 [==============================] - 0s 131ms/step
[7.207395e-05, 0.99502385, 0.0001242527, 0.0010733617, 0.00043210917, 0.00080536795,
0.0024689522] 1
1/1 [==============================] - 0s 107ms/step
1/1 [==============================] - 0s 147ms/step
1/1 [==============================] - 0s 121ms/step
1/1 [==============================] - 0s 138ms/step
```

10.7:Output For Yes Sign

CHAPTER 11

FUTURE EVALUATION

The future of sign language recognition holds exciting potential, driven by advancements in technology and machine learning. Here are some potential directions for its evolution:

1. Improved Accuracy: As machine learning algorithms become more sophisticated and datasets for sign language recognition grow, we can expect significant improvements in accuracy. This could involve leveraging deep learning techniques such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs) to better understand the nuances of sign language gestures.

2. Real-time Recognition: Future systems may aim for real-time recognition of sign language, allowing for seamless communication between sign language users and non-signers. This could involve the development of faster algorithms and more efficient hardware to process video inputs in real-time.

3. Gesture Variability: Sign language involves a wide range of gestures that can vary in appearance based on factors such as regional dialects, individual differences, and context. Future systems will need to account for this variability and develop robust models capable of recognizing a diverse range of gestures accurately.

4. Multi-modal Approaches: Combining multiple modalities such as video, depth sensors, and wearable devices could enhance the accuracy and reliability of sign

language recognition systems. For example, depth sensors can capture the 3D motion of gestures, providing additional information for recognition algorithms.

5. User Feedback Integration: Integrating feedback from sign language users into the development process can help improve the usability and effectiveness of recognition systems. This could involve techniques such as active learning, where the system actively seeks feedback from users to improve its performance over time.

6. Accessibility Integration: Future sign language recognition systems may be seamlessly integrated into various applications and devices to improve accessibility for sign language users. For example, they could be integrated into video conferencing platforms, virtual assistants, and educational tools to facilitate communication and learning.

Privacy and Ethical Considerations: As with any technology involving personal data, ensuring user privacy and addressing ethical concerns will be crucial. Future developments in sign language recognition should prioritize user consent, data security, and transparency in how data is collected and used.

Overall, the future of sign language recognition holds immense promise for improving communication accessibility for deaf and hard-of-hearing individuals, with advancements driven by ongoing research and collaboration between experts in machine learning, computer vision, and sign language linguistics.

CHAPTER 12
CONCLUSION

The "Real-Time Sign Language Recognition using CNN" project presents a promising foundation for future work and enhancements. Firstly, expanding the dataset to include a broader range of sign language gestures and variations would improve the model's accuracy and generalization ability. Additionally, integrating features such as facial expressions and body movements could enhance the richness and expressiveness of the communication interface. Moreover, exploring advanced deep learning techniques, such as recurrent neural networks (RNNs) or attention mechanisms, could further improve the system's performance, particularly in capturing temporal dependencies within sign language sequences. Furthermore, extending the application beyond text or speech translation to include real-time sign language generation could enable bidirectional communication between sign language users and non-users. Lastly, collaborating with sign language experts and communities to gather feedback and insights would ensure the system's relevance and effectiveness in real-world settings, fostering continuous improvement and innovation in sign language recognition technology.

SOURCE CODE

Data Collection Module:

```
import cv2
from cvzone.HandTrackingModule import HandDetector
import numpy as np
import math
import time
import os

# Initialize webcam
cap = cv2.VideoCapture(0)
detector = HandDetector(maxHands=1)  # Initialize hand detector
offset = 20  # Offset for bounding box
imgSize = 300  # Size for saving the image
counter = 0  # To count the number of images saved

# Folder to save the data images
folder = "Data/No"

# Creating the folder if it doesn't exist
if not os.path.exists(folder):
    os.makedirs(folder)

# Function to save image with unique name based on current timestamp
def save_image(img):
    global counter
    counter += 1
    timestamp = time.time()
    filename = f"{folder}/Image_{timestamp}.jpg"
    cv2.imwrite(filename, img)
    print(f"[INFO] Saved: {filename}")

while True:
    success, img = cap.read()  # Capture frame-by-frame
    hands, img = detector.findHands(img)  # Detect hands in the image

    if hands:
        hand = hands[0]
        x, y, w, h = hand['bbox']

        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255  # White
background

        imgCrop = img[y-offset:y + h + offset, x-offset:x + w + offset]  # Crop hand
image

 # Resize cropped image to standard size
        aspectRatio = h / w
        if aspectRatio > 1:
            k = imgSize / h
            wCal = math.ceil(k * w)
            imgResize = cv2.resize(imgCrop, (wCal, imgSize))
            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))
            hGap = math.ceil((imgSize - hCal) / 2)
            imgWhite[hGap:hCal + hGap, :] = imgResize

    # Display the cropped hand and the final image with white background
        cv2.imshow('ImageCrop', imgCrop)
        cv2.imshow('ImageWhite', imgWhite)

    # Show the original image with the hand detected
    cv2.imshow('Image', img)

    # GUI to handle key events
    key = cv2.waitKey(1)
    if key == ord("s"):
        save_image(imgWhite)  # Save the image with white background
    elif key == ord("q"):
        break  # Quit the program if 'q' key is pressed

# Release the video capture and destroy all windows when done
cap.release()
cv2.destroyAllWindows()
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("y"):  # Press 'y' to save image for "Yes" class
    counter_yes += 1
    folder = "Data/Yes"
    cv2.imwrite(f'{folder}/Image_{counter_yes}.jpg', imgWhite)
    print(f"Saved image for 'Yes', total: {counter_yes}")
```

```
        elif key == ord("n"):  # Press 'n' to save image for "No" class
            counter_no += 1
            folder = "Data/No"
        cv2.imwrite(f'{folder}/Image_{counter_no}.jpg', imgWhite)
            print(f"Saved image for 'No', total: {counter_no}")
        elif key == 27:  # Press 'Esc' key to exit
            break

cap.release()
cv2.destroyAllWindows();
```

Train Module:

```
import os
import cv2
import numpy as np
from skimage.feature import local_binary_pattern, hog
from sklearn.decomposition import PCA
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder

# Assuming each image is resized to 128x128 pixels
img_width, img_height = 128, 128
target_size = (img_width, img_height)

# Function to load and preprocess image data
def load_and_preprocess_data(data_path):
images = []
labels = []
for label in os.listdir(data_path):
label_path = os.path.join(data_path, label)
if os.path.isdir(label_path):
class_label = label.lower()
for file in os.listdir(label_path):
file_path = os.path.join(label_path, file)
```

```
if file.lower().endswith(('.jpg', '.png')):
# Load image
img = cv2.imread(file_path)
img = cv2.resize(img, target_size)
images.append(img)
labels.append(class_label)
return np.array(images), np.array(labels)

# Function to extract PCA features
def extract_pca_features(images):
num_components = min(len(images), images[0].size)  # Ensure n_components is
valid
pca = PCA(n_components=num_components)
gray_images = [cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) for image in
images]
reshaped_images = np.array([image.reshape(-1) for image in gray_images])
pca_features = pca.fit_transform(reshaped_images)
return pca_features

# Function to extract LBP features
def extract_lbp_features(images):
lbp_radius = 3
lbp_points = 24
lbp_features = []
for image in images:
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
lbp_feature = local_binary_pattern(gray_image, lbp_points, lbp_radius,
method='uniform')
```

```
lbp_features.append(lbp_feature.flatten())
return np.array(lbp_features)

# Function to extract HOG features
def extract_hog_features(images):
hog_features = []
for image in images:
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
hog_feature = hog(gray_image, orientations=8, pixels_per_cell=(8, 8),
cells_per_block=(1, 1), block_norm='L2-Hys')
hog_features.append(hog_feature.flatten())
return np.array(hog_features)

# Define data path
data_path = 'D:\Sign-Language-detection-main\Sign-Language-detection-
main\Data'

# Load and preprocess images
images, labels = load_and_preprocess_data(data_path)

# Extract PCA features
pca_features = extract_pca_features(images)

# Extract LBP features
lbp_features = extract_lbp_features(images)

# Extract HOG features
hog_features = extract_hog_features(images)
```

```
# Concatenate features
features = np.concatenate((pca_features, lbp_features, hog_features), axis=1)

# Encode labels
label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(labels)
num_classes = len(label_encoder.classes_)

# Convert labels to one-hot encoding
encoded_labels = to_categorical(encoded_labels, num_classes)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, encoded_labels,
test_size=0.2, random_state=42)

# Build the model
model = Sequential([
Dense(512, activation='relu', input_shape=(features.shape[1],)),
Dropout(0.5),
Dense(num_classes, activation='softmax')])

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)
```

```
# Train the model
model.fit(X_train,   y_train,   validation_split=0.2,   epochs=20,   batch_size=32,
callbacks=[early_stopping])

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {test_accuracy * 100:.2f}%')

# Save the trained model and label encoder classes
model.save('trained_model.h5')
with open('label_encoder_classes.npy', 'wb') as f:
np.save(f, label_encoder.classes_)
from sklearn.svm import SVC

# Build the SVM model
svm_model = SVC(kernel='rbf', C=1.0, gamma='scale')
# Train the SVM model
svm_model.fit(X_train, y_train)
# Evaluate the SVM model on the test set
svm_test_accuracy = svm_model.score(X_test, y_test)
print(f'SVM Test Accuracy: {svm_test_accuracy * 100:.2f}%')
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten

# Reshape data for CNN
X_train_cnn = X_train.reshape(-1, img_width, img_height, 1)
X_test_cnn = X_test.reshape(-1, img_width, img_height, 1)
```

```
# Build the CNN model
cnn_model = Sequential([
   Conv2D(32, (3, 3), activation='relu', input_shape=(img_width, img_height,1)),
   MaxPooling2D((2, 2)),
   Conv2D(64, (3, 3), activation='relu'),
   MaxPooling2D((2, 2)),
   Conv2D(64, (3, 3), activation='relu'),
   Flatten(),
   Dense(64, activation='relu'),
   Dense(num_classes, activation='softmax')])

# Compile the CNN model
cnn_model.compile(optimizer='adam',
loss='categorical_crossentropy',
             metrics=['accuracy'])

# Train the CNN model
cnn_model.fit(X_train_cnn,        y_train,        epochs=10,        batch_size=32,
validation_split=0.2, callbacks=[early_stopping])

# Evaluate the CNN model on the test set
cnn_test_loss, cnn_test_accuracy = cnn_model.evaluate(X_test_cnn, y_test)
print(f'CNN Test Accuracy: {cnn_test_accuracy * 100:.2f}%')
from sklearn.neighbors import KNeighborsClassifier

# Build the kNN model
knn_model = KNeighborsClassifier(n_neighbors=5)
```

```
# Train the kNN model
knn_model.fit(X_train, y_train)

# Evaluate the kNN model on the test set
knn_test_accuracy = knn_model.score(X_test, y_test)
print(f'kNN Test Accuracy: {knn_test_accuracy * 100:.2f}%')
```

Test Module:

```
import cv2
from cvzone.HandTrackingModule import HandDetector
from cvzone.ClassificationModule import Classifier
import numpy as np
import math
try:
cap = cv2.VideoCapture(0)
detector = HandDetector(maxHands=1)
classifier = Classifier("Model/keras_model.h5" , "Model/labels.txt")
offset = 20
imgSize = 300
counter = 0

labels = ["Hello","Yes","I love you","Okay","Please","No","Thank you"]


while True:
success, img = cap.read()
imgOutput = img.copy()
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
prediction , index = classifier.getPrediction(imgWhite, draw= False)
print(prediction, index)

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)


cv2.rectangle(imgOutput,(x-offset,y-offset-70),(x -offset+400, y - offset+60-
50),(0,255,0),cv2.FILLED)
```

```
cv2.putText(imgOutput,labels[index],(x,y-
30),cv2.FONT_HERSHEY_COMPLEX,2,(0,0,0),2)
cv2.rectangle(imgOutput,(x-offset,y-offset),(x + w + offset, y+h +
offset),(0,255,0),4)

cv2.imshow('ImageCrop', imgCrop)
cv2.imshow('ImageWhite', imgWhite)

cv2.imshow('Image', imgOutput)
cv2.waitKey(1)
except KeyboardInterrupt:
print("Interrupted by user, stopping...")
```

Test Voice Module:

```
import cv2
from cvzone.HandTrackingModule import HandDetector
from cvzone.ClassificationModule import Classifier
import numpy as np
import math
import pyttsx3

# Initialize the speech engine
engine = pyttsx3.init()

# Try-except block to handle exceptions
try:
    # Video capture object for the default camera
    cap = cv2.VideoCapture(0)
    # HandDetector object initialization with maximum one hand
    detector = HandDetector(maxHands=1)
    # Classifier object initialization with trained model and label files
    classifier = Classifier("Model/keras_model.h5", "Model/labels.txt")
    # Margins for cropping the hand image
    offset = 20
    # Size for the square image to input into the classifier
    imgSize = 300

    # List of labels for classification output
    labels = ["Hello", "Yes", "I love you", "Okay", "Please", "No", "Thank you"]
```

```
# Infinite loop to continuously capture frames
while True:
    # Capture a frame
    success, img = cap.read()
    # If frame capture fails, break the loop
    if not success:
        break

    # Make a copy of the frame
    imgOutput = img.copy()
    # Find hands in the frame
    hands, img = detector.findHands(img)
    # If a hand is detected
    if hands:
        # Get the first hand
        hand = hands[0]
        # Get the bounding box coordinates
        x, y, w, h = hand['bbox']

        # Create a white background for placing the cropped hand
        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255

        # Crop the hand image with offset
        imgCrop = img[y - offset:y + h + offset, x - offset:x + w + offset]

        # Calculate the aspect ratio of the hand
        aspectRatio = h / w
```

```
# If the hand is vertically longer
if aspectRatio > 1:
    # Scale according to height
    k = imgSize / h
    wCal = math.ceil(k * w)
    # Resize the cropped hand image
    imgResize = cv2.resize(imgCrop, (wCal, imgSize))
    # Calculate the horizontal gap for centering the hand on the white
background
    wGap = math.ceil((imgSize - wCal) / 2)
    # Place the resized hand in the center of the white background
    imgWhite[:, wGap: wCal + wGap] = imgResize

else:
    # Scale according to width
    k = imgSize / w
    hCal = math.ceil(k * h)
    # Resize the cropped hand image
    imgResize = cv2.resize(imgCrop, (imgSize, hCal))
    # Calculate the vertical gap for centering the hand on the white
background
    hGap = math.ceil((imgSize - hCal) / 2)
    # Place the resized hand in the center of the white background
    imgWhite[hGap: hCal + hGap, :] = imgResize

# Get the prediction and index from the classifier
prediction, index = classifier.getPrediction(imgWhite, draw=False)
```

```
            # Draw the label and bounding box on the output image
            cv2.rectangle(imgOutput, (x - offset, y - offset - 70), (x - offset + 400, y - offset + 60 - 50), (0, 255, 0), cv2.FILLED)
            cv2.putText(imgOutput, labels[index], (x, y - 30), cv2.FONT_HERSHEY_COMPLEX, 2, (0, 0, 0), 2)
            cv2.rectangle(imgOutput, (x - offset, y - offset), (x + w + offset, y + h + offset), (0, 255, 0), 4)

            # Display the cropped and the white background images
            cv2.imshow('ImageCrop', imgCrop)
            cv2.imshow('ImageWhite', imgWhite)

            # Use the text-to-speech engine to say the predicted label
            engine.say(labels[index])
            engine.runAndWait()

        # Display the output image
        cv2.imshow('Image', imgOutput)
        # Check for 'q' key to quit
        key = cv2.waitKey(1) & 0xFF
        if key == ord('q'):
            break
except Exception as e:
    # Print any exceptions to the console
    print(f"An error occus)
```

REFERENCES

[1]Singh, A. Wadhawan, M. Rakhra, U. Mittal, A. A. Ahdal and S. K. Jha, *2*022 10th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 2022, pp.

[2]M. Mohandes, M. Deriche and J. Liu. (2021) IEEE transactions on human-machine systems, vol. 44, p. 551–557.

[3]G. M. Rao, C. Sowmya, D. Mamatha, P. A. Sujasri, S. Anitha and R. Alivela, 2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS),Published on IEEEE transaction , Madurai, India, 2023, pp. 1086-1091

[4]https//ieeexplore.ieee.org/document/8537248

[5]A Practical preface to Computer Vision with OpenCV- WILEY nonstop dynamic Indian subscribe Language gesture recognition with steady backgrounds by Kumud Tripathi, Neha Baranwal,G.C. Nandi at 2015 Conference on Advances in Computing, Dispatches, and Informatics(ICACCI).

[6]https//towardsdatascience.com/lstm-recurrent-neural-networks-how-to-teach-a-network-toremember- the- history- 55e54

[7]H. Y. Lai and H. J. Lai, "Real-Time Dynamic Hand GestureRecognition," IEEE Int. Symp. Comput. Consum. Control,2014no. 1, pp. 658-661.

[8]Maeda Y, Wakamura M 2005 contemporaneous anxiety literacy rule for intermittent neural net- workshop and its FPGA perpetration IEEE Trans. Neural Network 16 6 1664 – 1672..

[9]Gallaudet University Research Institute A Detail Summary of Estimates for the Size of the Deaf Populatio

[10]S. Wei,X. Chen,X. Yang,S. Cao, andX. Zhang," A element- grounded vocabulary- expandible sign language recognition frame," Detectors,vol. 16,no. 4,p. 556, 2016.

[11]S. Kim,J. Kim,S. Ahn, andY. Kim, “ Finger language recognition grounded on ensemble artificial neural network literacy using armband emg detectors, ” Technology and Health Care,vol. 26, no. S1,pp. 249 – 258, 2018.
[12]M. De Coster,M. Van Herreweghe, andJ. Dambre, “insulated sign recognition from rgb videotape using disguise inflow and tone- attention, ” in Proceedings of the IEEE/ CVF Conference on Computer Vi- sion and Pattern Recognition, 2021,pp. 3441 – 3450.
[13]S. Jiang,B. Sun,L. Wang,Y. Bai,K. Li, andY. Fu, “Skeleton apprehensivemulti-modal sign lan- guage recognition, ” in Proceedings of the IEEE/ CVF Conference on Computer Vision and Pattern Recognition, 2021,pp. 3413 – 3423.
[14]I. Papastratis,K. Dimitropoulos, andP. Daras, “nonstop sign language recognition through a environment- apprehensive generative inimical network, ” Detectors,vol. 21,no. 7,p. 2437, 2021.(144)Y. Min,A. Hao,X. Chai, andX. Chen, “Visual alignment constraint for nonstop sign language recognition,” arXiv preprint arXiv2104.02330, 2021.
[15]S. Jiang,B. Sun,L. Wang,Y. Bai,K. Li, andY. Fu, “subscribe language recognition via shell- apprehensive multimodel ensemble, ” arXiv preprint arXiv2110.06161, 2021. [17] L. Meng and R. Li, “An attention-enhanced multi-scale and dual sign language recognition network based on a graph convolution network,” Sensors, vol. 21, no. 4, p. 1120, 2021.
[16]M. Boha´cek and M. Hr ˇ uz, “subscribe disguise- grounded trans- ´ former for word- position sign language recognition, ” in Proceedings of the IEEE/ CVF Winter Conference on operations of Computer Vi- sion, 2022, pp. 182 – 191
[17] “SIGN LANGUAGE RECOGNITION” Arugula Sismai 1, Asritha Diddi , Chavali Anusha 3, Chinta Sundara Sreya * 4, Prof.B. Prajna 5 International Research Journal of Modernization in En- gineering Technology and Science Volume04/ Issue05/ May- 2022, Impact Factor-6.752

[18]Shukor AZ, Miskon MF, Jamaluddin MH, Bin Ali F, Asyraf MF, Bin Bahar MB. 2015. A new data glove approach for Malaysian sign language det ection. Procedia Comput Sci 7660 – 67

[19]E.K. Kumar,P.V.V. Kishore,M.T.K. Kumar, andD.A. Kumar, “ 3D sign language recognition with common distance and angular enciphered color topographical descriptor on a 2 – sluice CNN, ” Neurocomputing, vol. 372, pp. 40 – 54, Jan. 2020

[20]Breland,D.S., Skriubakken,S.B., Dayal,A., Jha,A., Yalavarthy,P.K. and Cenkeramaddi,L.R., Deep Learning- Grounded subscribe Language integers Recognition From Thermal Images With Edge Computing System. IEEE Sensors Journal, vol. 21(9), pp.10445- 10453, 2021

[21]Nearest neighbour classification of Indian sign language gestures using kinect camera, Z. A. Ansari and G. Harit. Sadhana, IEEE transactions vol. 41, p. 161–182(2021)

[22]A new data glove approach for Malaysian sign language detection, Z.Shukor, M. F.Miskon, M. H. Jamaluddin, F. bin Ali, M. F. Asyraf, M. B. bin Bahar and others. (2020)

[23]Image-based and sensor-based approaches to Arabic sign language recognition, M. Mohandes, M. Deriche and J. Liu. ,IEEE transactions on human-machine systems, vol. 44, p. 551–557. (2021)