MAJOR PROJECT REPORT

GESTURE GENIE (ASL - SPEECH CONVERSION)

Submitted in partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING

Submitted by

Aditi Malik	**University Roll No. :** 2118125
Stuti Mishra	**University Roll No. :** 2119281
Suraj Chauhan	**University Roll No. :** 2119309

Under the guidance of

Dr. Vikrant Sharma

*Assistant Professor (*Department of Computer Science and Engineering, Graphic Era Hill University, Dehradun.*)*

Project Group No: 211

Department of Computer Science and Engineering

Graphic Era Hill UniversityJune, 2025

Graphic Era
HILL UNIVERSITY
Established by an Act of the State Legislature of Uttarakhand (Adhiniyam Sankhya 12 of 2011)
University under section 2(f) of UGC Act, 1956

CANDIDATE'S DECLARATION

We hereby certify that the work which is being presented in the project progress report entitled **"GESTURE GENIE (ASL - SPEECH CONVERSION)"** in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering in the Department of Computer Science and Engineering of the Graphic Era Hill University, Dehradun shall be carried out by the undersigned under the supervision of **Dr.Vikrant Sharma**,**Assistant Professor**, Department of Computer Science and Engineering, Graphic Era Hill University, Dehradun.

Aditi Malik	University Roll No. : 2118125
Stuti Mishra	University Roll No. : 2119281
Suraj Chauhan	University Roll No. : 2119309

The above mentioned students shall be working under the supervision of the undersigned on the **"GESTURE GENIE (ASL - SPEECH CONVERSION)"**

Guide : Dr. Vikrant Sharma

Date : 20th May 2025

Place : Graphic Era Hill University , Dehradun

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to the individuals and institutions that have supported the development of this Sign Language to Gesture Genie project.

First and foremost, we acknowledge the invaluable contributions of the deaf and hard-of-hearing community. Their insights and feedback have been crucial in shaping the direction and focus of this research, ensuring its relevance and potential impact.

We extend our heartfelt thanks to **Dr. Vikrant Sharma** for their unwavering guidance, insightful feedback, and continuous encouragement throughout the course of this project. Their expertise in deep learning has been instrumental in navigating the complexities of this endeavor.

We are also grateful to Graphic Era Hill University for providing the necessary resources, infrastructure, and a conducive environment that facilitated this research. The access to computational facilities, datasets, libraries that has been essential for the successful implementation of the Keras-based models and the overall system development

This project would not have been possible without the collective support and contributions of these individuals and entities. We are deeply appreciative of their involvement and belief in the potential of this work to foster more inclusive communication.

Aditi Malik University Roll No. : 2118125

Stuti Mishra University Roll No. : 2119281

Suraj Chauhan University Roll No. : 2119309

ABSTRACT

Gesture Genie: Sign to Speech Translator is an assistive technology project designed to bridge the communication gap between the deaf/mute community and the hearing population. The system leverages computer vision and deep learning techniques to recognize American Sign Language (ASL) gestures and convert them into real-time speech output. Using a convolutional neural network (CNN) trained on hand landmark data extracted via Mediapipe, the model identifies static signs corresponding to alphabets and translates them into audible words through a user-friendly desktop interface. The solution provides high accuracy, swift response time, and practical usability for real-world scenarios. Additionally, the project proposes future integration of Natural Language Processing (NLP) features, such as next-word prediction, grammar correction, and common phrase shortcuts, to make conversations more fluid and natural. These advancements aim to transform the translator from a basic gesture-to-speech system into a smart communication assistant. The project has significant social impact potential, offering an inclusive technological solution that enhances independence and self-expression for the hearing-impaired community. Its scalable architecture also supports future developments across platforms like mobile and web applications. *Gesture Genie* represents a meaningful step toward inclusive, AI-powered communication for all.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Communication is the bedrock of human interaction, serving as the fundamental conduit through which we exchange thoughts, emotions, ideas, needs, and experiences. For the vast majority of people, this essential function is facilitated by spoken language, often enriched by a symphony of facial expressions, gestures, and body language. In our daily lives, dynamic conversations unfold effortlessly – voices rise and fall, replies are heard, responses are given, interruptions occur, and a constant flow of information circulates. Spoken language has, by default, been the primary mode of communication across societies worldwide, from the intimate settings of homes and bustling cafes to the more formal environments of schools, hospitals, and workplaces. Beyond the spoken word, we increasingly rely on written text and, more recently, digital mediums such as messaging apps, emails, and voice notes to maintain connections. However, for a significant segment of the global population, specifically individuals who are unable to speak or hear—commonly referred to as deaf and mute individuals—these prevalent methods present profound and often isolating challenges.

Deaf and mute individuals are those who either cannot hear (deaf), cannot speak (mute), or frequently, both. While many are born with these impairments, others may acquire them due to accidents, medical conditions, or the natural process of aging. The statistics underscore the gravity of this global issue: the World Health Organization (WHO) estimates that over 430 million people worldwide currently experience disabling hearing loss. This alarming figure is projected to surge to over 700 million by 2050. India, a nation with a vast population, is no exception, with approximately 18 million people—exceeding 1% of its total population—estimated to be deaf or hard of hearing. A notable subset within this community is also non-verbal, meaning they cannot produce intelligible speech. These individuals navigate their daily lives by relying on alternative communication strategies, including lip-reading, written text, facial expressions, and, most crucially, sign language.

Sign language stands as a sophisticated and fully developed system of communication, employing a rich lexicon of hand gestures, facial expressions, and precise body movements to convey meaning. It possesses its own distinct grammar and expansive vocabulary, much like any spoken language. It's important to recognize that sign languages are not universal; rather, they are diverse and regionally specific. For instance, American Sign Language (ASL), British Sign Language (BSL), and Indian Sign Language (ISL) are unique linguistic systems, each distinct from one another and from their corresponding spoken languages. Within deaf communities, sign language is not merely an alternative; it is the primary and most natural mode of communication. These communities often exhibit remarkable fluency in sign language, fostering tight-knit bonds and shared cultural and linguistic norms.

Sign language is a structured system of communication using hand gestures, facial expressions, and body movements to convey meaning. It is a fully developed language with its own grammar and vocabulary. Various regions and countries have their own sign languages—such as American Sign Language (ASL), British Sign Language (BSL), and Indian Sign Language (ISL)—which are distinct from one another and from spoken languages. Among deaf communities, sign language is the primary and most natural mode of communication. They use it fluently to communicate with each other, often forming tight-knit communities with shared cultural and linguistic norms.

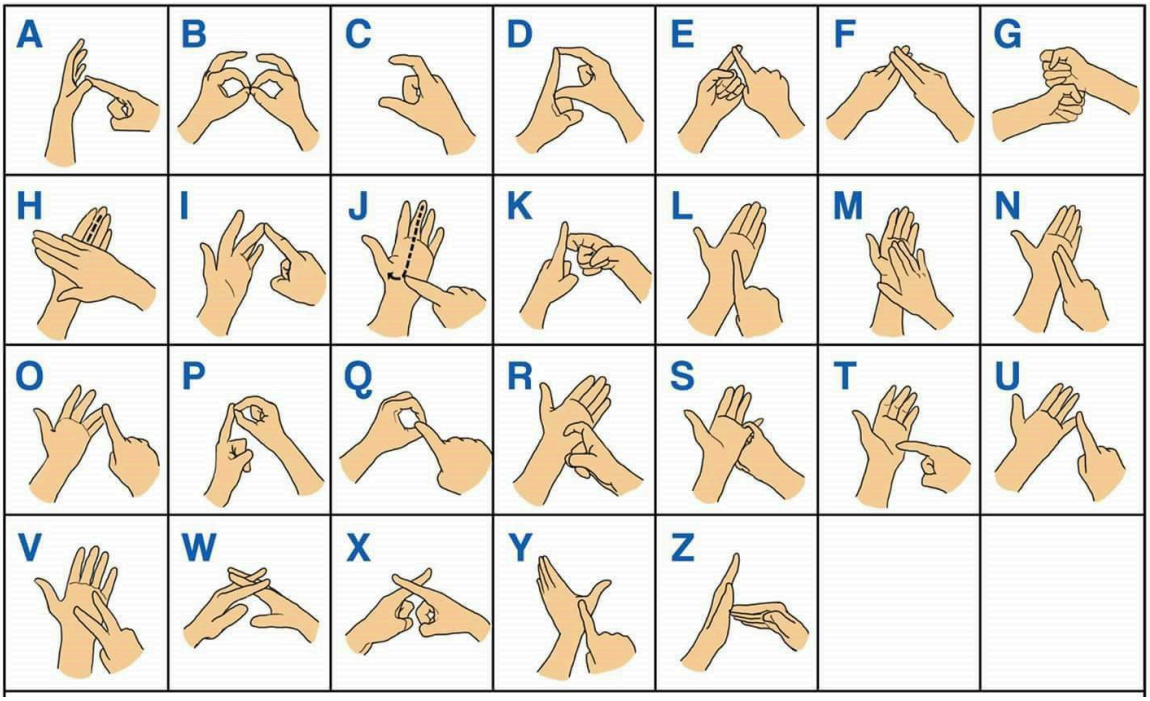

Fig. 1.1 BSL(British Sign Language)

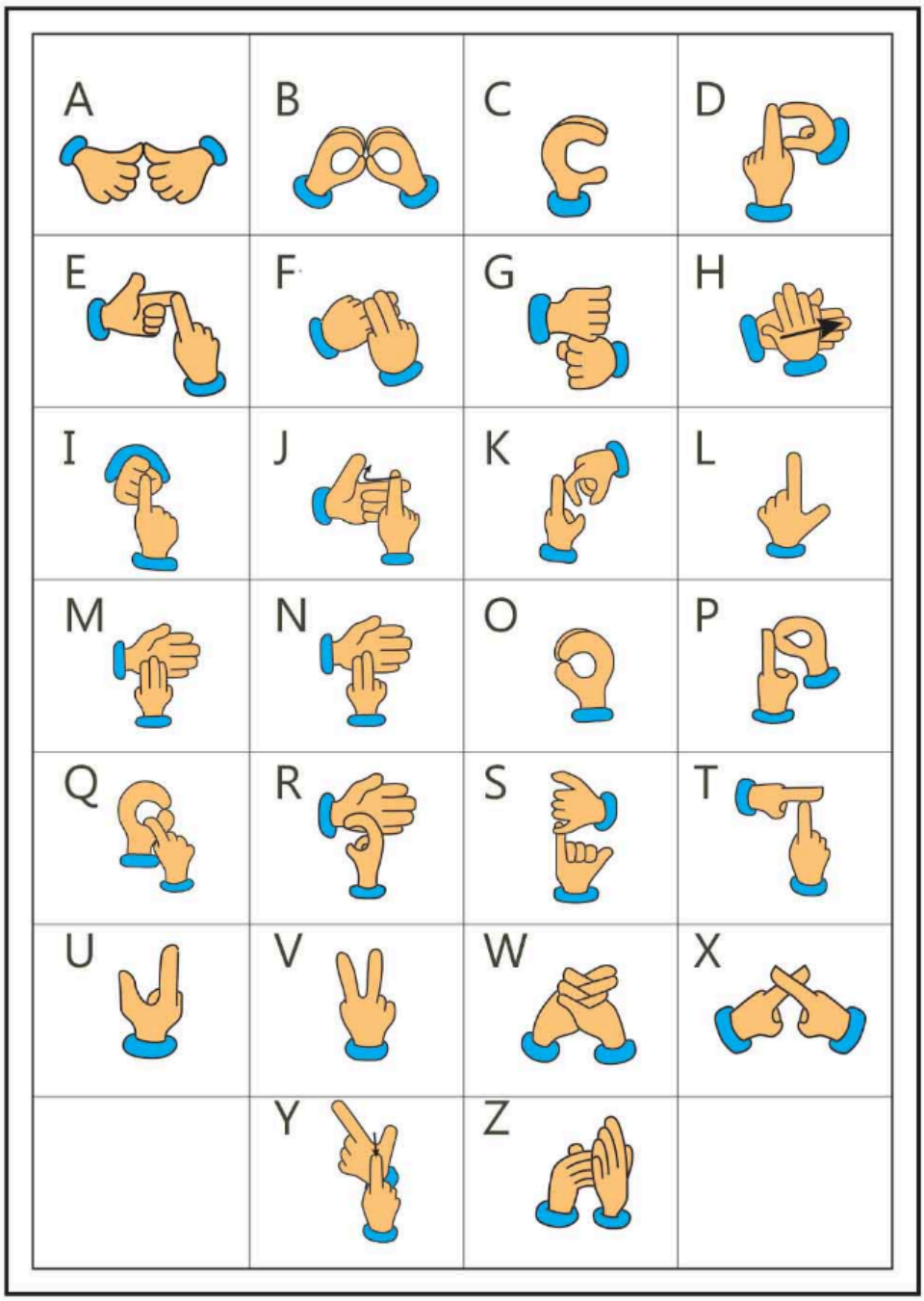

Fig. 1.2 ISL (Indian Sign Language)

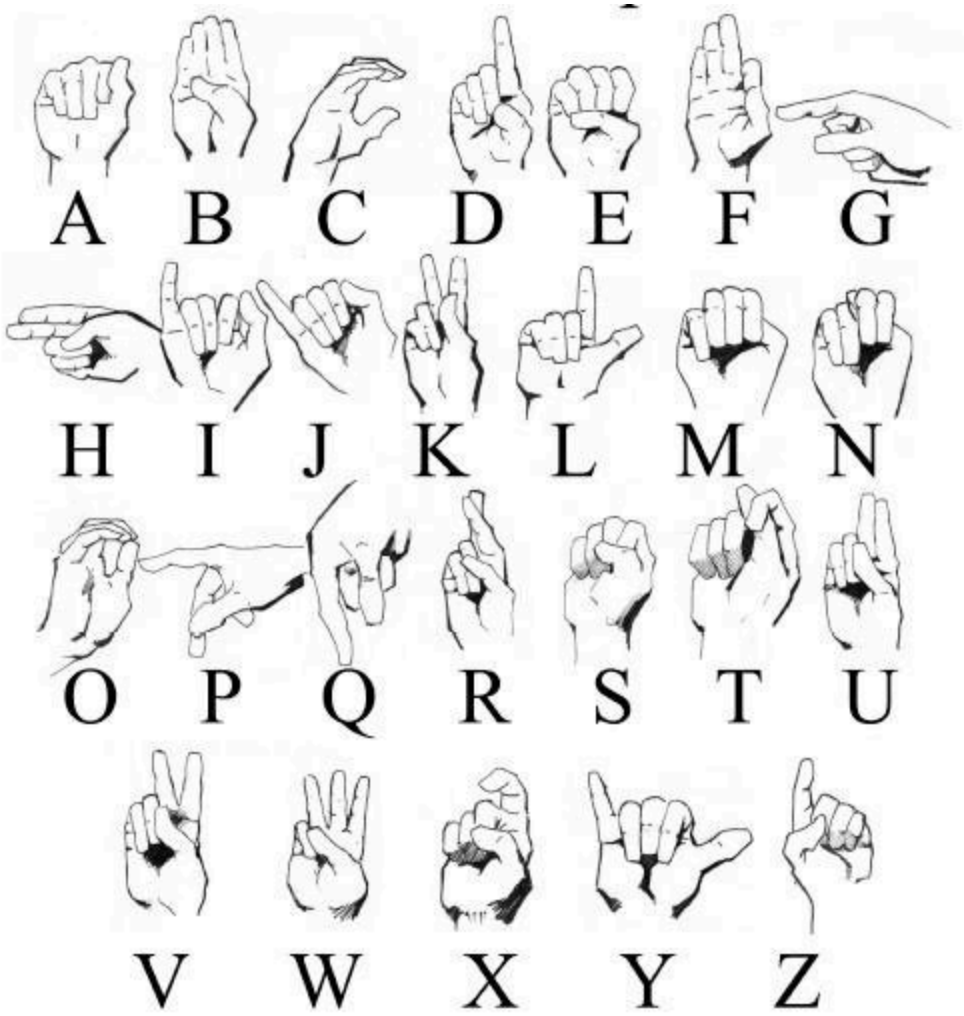

Fig. 1.3 ASL(American Sign Language)

Despite the richness and efficacy of sign language within its communities, significant challenges emerge when deaf or mute individuals attempt to communicate with the majority of the population who do not understand it. This widespread lack of sign language proficiency creates a profound communication barrier. In their daily lives, deaf and mute individuals often struggle to articulate urgent needs, express complex thoughts, or fully participate in social interactions. Tasks that most people take for granted—such as visiting a doctor, seeking assistance in a retail store, or applying for a job—can become overwhelmingly difficult. They are frequently compelled to resort to writing

notes or employing ambiguous gestures, leading to misunderstandings, acute frustration, and a pervasive sense of social isolation.

This pervasive communication gap has far-reaching consequences. It not only restricts personal and professional opportunities but also severely impedes their access to essential services and their full integration into society. In emergency situations, the inability to communicate swiftly and clearly can even escalate to a life-threatening scenario. Therefore, there is an undeniable and urgent need for innovative solutions that can effectively bridge this divide, fostering genuinely inclusive communication between hearing and non-hearing individuals.

Enter the transformative potential of modern technology, particularly the convergence of computer vision, machine learning, and natural language processing. By harnessing the power of cameras to detect intricate hand gestures and sophisticated algorithms to interpret them, we can develop systems capable of translating sign language into spoken words or written text in real-time. This technological breakthrough holds the promise of acting as a personal translator or interpreter, empowering users to "speak" to others through their gestures. Such systems are not merely functional tools; they represent a beacon of hope for fostering greater independence and self-expression among deaf and mute individuals, while simultaneously promoting awareness and inclusion across society.

One promising solution lies in the integration of modern technology—particularly computer vision, machine learning, and natural language processing. By using cameras to detect hand gestures and algorithms to interpret them, we can build systems that translate sign language into spoken words or text in real time. This technology has the potential to act as a translator or interpreter for the user, allowing them to "speak" to others through their gestures. Such systems can not only foster independence and self-expression among deaf and mute individuals but also promote awareness and inclusion in society.

It is within this pressing context that our project, "Gesture Genie: Sign to Speech Translator," emerges as a promising solution. We have meticulously developed an AI-powered desktop application engineered for the real-time translation of American

Sign Language (ASL) gestures into audible speech. Our core objective is to empower non-verbal individuals, enabling them to communicate more effectively and fluidly with those who do not possess an understanding of sign language. The system is designed with an unwavering focus on simplicity, accessibility, and efficiency, aiming to serve as a digital bridge in everyday interactions.

The operational essence of Gesture Genie lies in its sophisticated gesture recognition capabilities, driven by advanced computer vision. We leveraged Mediapipe, an open-source framework meticulously developed by Google, for its unparalleled ability to detect and track hand landmarks. These landmarks—critical points on the hand that precisely describe its posture and position—are the raw data for our system. This visual data is then meticulously processed by a Convolutional Neural Network (CNN), chosen for its inherent strength in image processing and pattern recognition. The CNN is trained to accurately classify these hand gestures into their corresponding alphabets or words. Upon successful recognition, the classified gesture is immediately routed to a text-to-speech engine, which vocalizes the gesture as a spoken word. This audible output, delivered through speakers, ensures that the user's message is instantly comprehensible to nearby hearing individuals. The user interface of Gesture Genie is intuitively designed, featuring clear instructions and a real-time preview of the hand gestures, their predicted classifications, and the resultant spoken output, providing immediate feedback to the user.

The core of our system involves gesture recognition using computer vision. We utilized Mediapipe, an open-source framework by Google, to detect hand landmarks—key points on the hand that describe its posture and position. These landmarks are then processed using a Convolutional Neural Network (CNN), which classifies the hand gestures into corresponding alphabets or words. We chose CNN because of its strength in image processing and pattern recognition. The model was trained on a dataset of ASL gestures and tested for accuracy and performance.

Once a gesture is recognized, it is passed to a text-to-speech engine that vocalizes the gesture as a spoken word. The speech output can be heard through speakers, making it easy for people nearby to understand what the user is trying to communicate. The user

interface is designed to be intuitive, with clear instructions and a real-time preview of hand gestures, predictions, and outputs.

Additionally, we plan to integrate NLP-powered features in the future to enhance the naturalness of conversation. These include:

- Next-word prediction, which will help suggest what word might come next based on the current context, speeding up communication.
- Grammar correction and auto-capitalization, which will refine the output to make it more readable and professional.
- Common phrase shortcuts, which will allow users to quickly express frequently used sentences like "Thank you" or "I need help".

The scope of this project is vast. While our current implementation supports static ASL alphabets, we plan to expand it to dynamic gestures and full sentences in the future. The technology can also be adapted to mobile apps, wearables, or smart glasses, making it even more portable and accessible. This project not only demonstrates the practical application of AI in solving real-world problems but also underscores the role of inclusive design in technology. By giving a voice to those who cannot speak, we take a step forward toward a more empathetic, connected, and equal society.

This is where Gesture Genie comes into play—an innovative AI-powered desktop application designed to translate sign language into audible speech using computer vision and machine learning. The goal of the project is to empower non-verbal individuals by enabling real-time, two-way communication with people who do not understand sign language. The application focuses on gesture recognition and speech synthesis, turning hand signs into spoken words through an intuitive user interface.

From a technical perspective, Gesture Genie integrates several advanced tools and frameworks to make this possible:

From a technical vantage point, Gesture Genie meticulously integrates several cutting-edge tools and frameworks to realize its vision:

1. **Mediapipe for Real-Time Hand Tracking**

   At the foundation of our system is Google's Mediapipe framework, specifically utilized for its robust real-time hand tracking and gesture landmark detection capabilities. Mediapipe offers a highly optimized pipeline, capable of detecting 21 key landmark points on each hand from a single camera frame. These landmarks, which include fingertips, knuckles, and the base of the hand, provide an incredibly detailed skeletal map of the hand's posture. This intelligent pre-processing dramatically streamlines the complexity of handling raw image data, making the system exceptionally efficient for real-time applications.

2. **Feature Extraction and Normalization**

   Once the 21 landmark points are accurately extracted, their coordinates undergo a crucial normalization process (scaled between 0 and 1). Furthermore, their relative positions are meticulously calculated. This vital preprocessing step guarantees consistency in the input features, irrespective of variations in hand size or camera distance. These normalized vectors then serve as the precise input features for the subsequent stage: gesture classification.

3. **CNN-Based Gesture Classification**

   For the critical task of gesture classification, we engineered a sophisticated Convolutional Neural Network (CNN) leveraging the powerful TensorFlow and Keras libraries. This CNN was rigorously trained on a custom-curated dataset comprising labeled hand gestures corresponding to ASL alphabets and frequently used common words. Our dataset deliberately includes a diverse mix of static gesture images (covering alphabets A-Z and common signs like "Hello," "Yes," "No," etc.), captured under a variety of lighting and background conditions to significantly enhance the model's robustness and generalization capabilities. The CNN architecture thoughtfully incorporates multiple convolutional layers, pooling layers, and dense layers to effectively learn the intricate spatial patterns inherent in various hand shapes. The final output layer employs a softmax classifier, which confidently predicts the most probable gesture class. During validation, the CNN

achieved an impressive over 95% accuracy for static gestures, affirming its reliability.

4. **Speech Synthesis (Text-to-Speech)**

Upon the successful classification of a gesture, its corresponding word is seamlessly transmitted to a text-to-speech (TTS) engine. For this purpose, we selected pyttsx3, a versatile Python library that functions offline and efficiently converts text into spoken audio. This integration ensures that the user's gestures are instantly vocalized, providing a smooth and natural way for them to "speak" using their sign language.

5. **Real-Time GUI using Tkinter and OpenCV**

To maximize user interaction and accessibility, we developed a user-friendly graphical interface (GUI) for our desktop application using Tkinter. This interface prominently features a live video feed (powered by OpenCV), displaying the user's hand in real-time. Alongside the video, the GUI presents the predicted gesture and the corresponding spoken output. This immediate visual feedback allows users to observe how the system is interpreting their signs and adjust accordingly. The GUI also incorporates intuitive controls, including buttons to initiate or halt the prediction process and to toggle the voice output.

6. **Performance Optimization**

Real-time systems demand meticulous performance optimization. To achieve this, we ingeniously employed threading in Python, enabling parallel processing of video input and model prediction. This parallel execution significantly reduces latency and ensures exceptional responsiveness. Furthermore, our model is designed to be lightweight, allowing it to run smoothly on most standard systems without necessitating high-end GPUs.

7. **Dataset Handling and Training**

   Our robust dataset encompasses thousands of meticulously labeled gesture samples spanning multiple classes (alphabets and words). We rigorously split this data into dedicated training and validation sets, applying advanced data augmentation techniques such as mirroring and rotation to enhance the model's learning capacity and resistance to overfitting. Throughout the training process, we diligently monitored model accuracy across epochs to ensure optimal performance and generalization.

8. **Modular Architecture**

   The project's design adheres to a modular architecture, with distinct and independent components for data capture, model training, real-time prediction, and speech generation. This modularity ensures unparalleled flexibility and ease of future upgrades. This includes the seamless addition of more gestures (whether dynamic or contextual), the ability to switch between different TTS engines, potential deployment as a web or mobile application, and the future integration of sophisticated NLP models for enhanced sentence formation and grammar correction.

Through Gesture Genie, we demonstrate how modern technologies can be harnessed to create meaningful social impact. The system serves as a digital interpreter, enabling deaf and mute individuals to have a voice in real time, reducing their dependence on others, and promoting dignity, inclusion, and autonomy.

Our long-term vision includes expanding the project into a mobile app and incorporating Natural Language Processing (NLP) capabilities. Planned NLP-powered features include:

- Next-word prediction: Using models like GPT or BERT, the system can suggest words based on the user's previous gestures, allowing for faster sentence construction.
- Grammar correction and auto-capitalization: Ensuring that the translated message is polished and professional.

- Common phrase shortcuts: Automatically recognizing frequently used ASL phrases like “How are you?”, “I need help”, or “Thank you” and replacing them with full spoken sentences.

In summary, Gesture Genie is a step toward an inclusive future, where communication is not limited by physical ability. It combines computer vision, deep learning, NLP, and real-time speech synthesis into a single, accessible platform. By giving a digital voice to hand gestures, this project enhances the quality of life for the deaf and mute community and bridges a vital gap between different segments of society.

CHAPTER 2

OBJECTIVES OR PROBLEM STATEMENT

2.1 Problem Statement

Communication is a fundamental human right, yet millions of individuals who are deaf or hard of hearing face daily challenges when trying to convey their thoughts and emotions to those unfamiliar with sign language. This difficulty creates significant barriers in their daily lives. Among the many forms of non-verbal communication, American Sign Language (ASL) is one of the most widely used, particularly across North America. It's a rich, complex visual language, but despite its prevalence, a significant portion of the global population does not understand or interpret ASL. This widespread unfamiliarity creates a persistent communication gap that severely hinders social inclusion, limits access to educational opportunities, and negatively impacts employment prospects for individuals who rely on it. The lack of mutual understanding can lead to frustration, isolation, and a reduced quality of life for deaf and hard of hearing individuals.

2.1.1 Limitations of Current Solutions

The traditional methods available for interpreting sign language often involve human interpreters or reliance on expensive, specialized proprietary hardware. While human interpreters provide invaluable services, they are inherently not scalable to meet the vast daily communication needs of the entire deaf and hard of hearing community. Their availability can be limited, especially in spontaneous or emergency situations, and their services are often not economically feasible for everyone. This creates an accessibility bottleneck, particularly in less formal settings or in areas with limited resources.

Furthermore, many existing sign language recognition systems fall short of practical utility. They are frequently not real-time, meaning there's a noticeable delay between a sign being made and its interpretation, which disrupts natural conversation flow. Others

lack accessibility, requiring high-end computational power or specialized sensors that put them out of reach for the average consumer. The unfortunate reality is that most public spaces, educational institutions, and even digital platforms are still not adequately equipped to facilitate smooth, immediate interactions with people who communicate using sign language. This systemic oversight further widens the gap in accessibility and inclusion.

2.1.2 The Need for Accessible Learning Tools

Another pressing issue compounding this problem is the lack of intuitive and interactive educational tools for those who wish to learn ASL. While there's a growing interest in learning sign language, students, educators, and enthusiasts often struggle to find engaging, feedback-driven platforms that allow them to learn and practice in real-time. Traditional learning methods, such as textbooks or pre-recorded videos, lack the dynamic interactivity necessary for mastering a visual language like ASL. This deficiency significantly limits the overall outreach and effectiveness of sign language education, perpetuating the cycle of limited understanding among the hearing population. Without accessible and effective learning resources, the ability to bridge the communication gap remains severely constrained.

2.1.3 Leveraging Modern Technology for Inclusion

However, with the rapid advent and maturation of artificial intelligence (AI) and computer vision, it has become technically feasible to recognize intricate hand gestures in real-time. This can now be achieved using widely available consumer-grade hardware, such as standard webcams, and by deploying lightweight deep learning models. This technological potential offers a glimmer of hope for a truly transformative solution. Despite this incredible promise, there remains a notable lack of open, interactive platforms that seamlessly integrate this cutting-edge technology with user-friendly interfaces, crucial audio feedback, and the ability for real-time word formation from sequential gesture input. Many existing prototypes are confined to research labs or lack the polish for widespread adoption.

The absence of such a comprehensive and accessible tool represents a significant missed opportunity for profoundly improving accessibility, dramatically enhancing learning opportunities, and genuinely fostering inclusion for a large and deserving demographic. Imagine the empowerment a system could provide if it could translate hand gestures into text and speech using accessible hardware and open-source tools. This would undoubtedly be a game-changer—not only for individuals with hearing impairments, granting them greater autonomy and voice, but also for their families, educators, healthcare providers, employers, and society at large, fostering a more connected and understanding world.

2.1.4 Addressing the Core Problem

Therefore, the core problem this project seeks to address lies in the critical absence of an accessible, real-time, cost-effective, and user-friendly hand gesture recognition system that can effectively:

- Detects and classifies ASL hand signs accurately from live video input.
- Convert recognized signs into both text and speech for seamless, multimodal communication.
- Allow dynamic, real-time interaction through an intuitive web interface, making it easy to form words and sentences.
- Be utilized effectively as both an assistive communication tool for daily interactions and an engaging educational platform for ASL learners.

This project is meticulously designed to bridge this crucial gap by building an integrated system that harnesses the power of computer vision, deep learning, Flask for web development, and speech synthesis to bring real-time ASL recognition and invaluable feedback directly to everyday users.

2.2 Objectives

The primary objective of this project is to design and implement a cutting-edge real-time American Sign Language (ASL) hand gesture recognition system, leveraging the power of computer vision and deep learning technologies. This innovative system is presented through an intuitive and interactive web interface, aiming to establish a crucial and meaningful bridge between the hearing and non-hearing communities. By facilitating the real-time translation of sign language into both readable text and audible spoken words, the project seeks to empower individuals and foster more inclusive communication environments.

The project integrates multiple core components, each meticulously designed to address a key objective and contribute to the system's overall functionality and effectiveness:

2.2.1 Hand Detection and Image Preprocessing

At the foundational level, the system begins with Hand Detection and Image Preprocessing. Utilizing robust libraries such as OpenCV and the specialized CVZone library, the system efficiently captures live video streams directly from the user’s webcam. The initial crucial step involves the precise identification of the hand region within the video frame using a dedicated hand detection module. This module is trained to accurately locate and isolate the human hand, distinguishing it from other elements in the background. Once the hand gesture is successfully detected, it undergoes a series of vital preprocessing steps: the detected hand region is meticulously cropped to focus solely on the gesture, then resized to a consistent dimension (specifically, 300x300 pixels), and finally normalized onto a pristine white background. This rigorous normalization process is paramount; it ensures that all input images fed to the deep learning model are uniform in size, scale, and background, thereby minimizing variability and significantly enhancing the consistency and accuracy of subsequent model predictions, regardless of the user's distance from the camera or varying lighting conditions. This standardization is critical for the model's ability to generalize across different users and environments.

2.2.2 Dataset Collection for Model Training

A cornerstone of any robust deep learning system is a comprehensive and high-quality dataset. Recognizing this, the project thoughtfully incorporates a dedicated module for Dataset Collection for Model Training. This functionality empowers users to actively participate in the system's improvement by capturing and securely storing their own hand gesture images in a predefined folder structure. This user-friendly feature is invaluable for several reasons: it allows for the continuous expansion or fine-tuning of the training dataset, particularly for specific or nuanced sign language gestures that might be underrepresented. Over time, as more diverse data is collected and integrated, the model's accuracy and adaptability are significantly enhanced. This iterative data collection process ensures the system remains relevant and increasingly precise, capable of recognizing a broader spectrum of ASL signs and adapting to individual signing styles, thereby fostering a truly collaborative and evolving learning environment for the AI.

Gesture Classification Using a Deep Learning Model

The heart of the system lies in its Gesture Classification Using a Deep Learning Model. A pre-trained Convolutional Neural Network (CNN) model, meticulously developed using the powerful TensorFlow/Keras frameworks and integrated seamlessly via a custom Classifier module, is responsible for the intricate task of recognizing alphabet hand signs from ASL. This CNN is a sophisticated neural network architecture specifically designed to process visual data, excelling at identifying complex patterns and features within images. In real-time, as the user performs a hand gesture in front of the webcam, the preprocessed image of the gesture is fed into this trained CNN. The model then rapidly analyzes the visual input, comparing it against the vast patterns it learned during its training phase, and predicts the most probable letter corresponding to the live hand gesture. This predicted letter is then instantly displayed on the video feed within the web interface, providing immediate visual feedback to the user. The accuracy and speed of this classification are critical for a fluid and natural communication experience.

2.2.3 Web-Based Real-Time Interface

To ensure accessibility and a user-friendly experience, a Flask-based web application serves as the dynamic front end of the system. Flask, a lightweight Python web framework, was chosen for its simplicity and efficiency in handling real-time data streams. This web interface is designed to stream the live video feed directly from the user's webcam, allowing them to view their gestures as the system processes them. Crucially, it dynamically displays the real-time predictions generated by the CNN model. Beyond mere display, the interface supports a range of essential user interactions: users can easily add the currently recognized letter to form a word, effectively building sentences one sign at a time. They can also delete the last entered letter in case of a misrecognition or error, add a space to separate words, or, most importantly, trigger the conversion of the full accumulated word into speech. This interactive design ensures that users have full control over the communication process, making it intuitive and responsive.

2.2.4 Text-to-Speech Feedback

To complete the communication loop and enable interaction with non-signers, the project seamlessly incorporates robust Text-to-Speech (TTS) Feedback. This vital component utilizes Google Text-to-Speech (gTTS), a powerful library that converts written text into natural-sounding speech, in conjunction with Pygame, which is used for audio playback. Once the deep learning model successfully recognizes and forms a complete word from a sequence of gestures, this word is passed to the gTTS engine. The engine then synthesizes the word into an audio format, which is immediately played back through the user's speakers via Pygame. This real-time audible output is incredibly practical in various everyday scenarios, such as ordering food in a restaurant, asking questions in a public setting, or simply expressing basic needs. The immediate spoken feedback transforms the silent act of signing into an audible message, effectively bridging the communication gap and fostering direct interaction.

2.2.5 Ease of Use and Accessibility

A fundamental design principle for this project was Ease of Use and Accessibility. The system is engineered to run effortlessly on standard, readily available hardware, such as typical laptops equipped with built-in webcams. Critically, it does not necessitate the use of expensive, specialized GPUs or proprietary sensors, which often act as significant barriers to adoption for assistive technologies. This deliberate design choice ensures that the system can be widely adopted and deployed in diverse settings, including educational institutions like schools, private residences, or public kiosks. By minimizing hardware requirements and maximizing compatibility, the project aims to make this transformative communication tool accessible to the broadest possible audience, truly democratizing access to assistive technology.

2.2.6 Expandability and Customization

The underlying code structure of the project is inherently modular, a design choice that champions Expandability and Customization. This thoughtful architecture allows for effortless future enhancements and adaptations. For instance, additional sign language gestures, a broader vocabulary of words, or even more complex dynamic gestures (which involve motion over time, unlike static signs) can be seamlessly incorporated into the system. This expansion would primarily involve retraining or replacing the current deep learning model with an updated version that includes the new data. This modularity ensures the project is not a static solution but a dynamic platform capable of evolving with the needs of the ASL community and advancements in AI research, promising long-term utility and relevance.

Overall, the objective of this project transcends merely building a functional sign-to-text system. It is fundamentally about creating an inclusive communication tool that is not only highly effective but also remarkably easy to use, expand, and adapt for a wide array of educational and assistive purposes. By strategically combining the cutting-edge capabilities of Artificial Intelligence, modern web development, and human-centered design principles, this project makes a significant and tangible contribution toward fostering a more inclusive and accessible society for everyone.

CHAPTER 3

LITERATURE SURVEY

Sign language recognition (SLR) has garnered significant attention as an essential technological advancement aimed at bridging persistent communication gaps between the deaf/mute community and hearing individuals. The development of effective SLR systems is crucial for fostering greater inclusivity and enabling seamless daily interactions. Over the years, researchers have proposed and explored various sophisticated approaches, predominantly leveraging cutting-edge machine learning and deep learning techniques. These methodologies have been applied to diverse sensor inputs, ranging from specialized data gloves to conventional cameras, and trained on a multitude of datasets, each with its own specific characteristics and challenges. This burgeoning field continues to evolve rapidly, driven by the increasing availability of computational power and advanced algorithms. This literature survey aims to provide a concise overview by summarizing five notable studies within this domain, highlighting their distinct methodologies, the types of datasets utilized for training and evaluation, and their reported performance metrics to illustrate the progress and remaining hurdles in real-time sign language interpretation.

1. This study integrates Random Forest algorithms with the Leap Motion Controller to detect real-time hand gestures, enabling accurate sign language recognition. The methodology includes preprocessing with MediaPipe for feature extraction, CNN-based training, and a Tkinter-built user interface. Pyttsx3 is used to convert recognized signs into speech, promoting inclusivity. The system's performance is evaluated using precision, recall, and F1-score, demonstrating promising accuracy for practical communication aids [1]. This work emphasizes the use of accessible hardware combined with robust machine learning models to facilitate real-time sign-to-speech conversion.
2. Addressing dynamic and complex gestures in ASL, this research utilizes a 3D CNN architecture to capture spatial-temporal features from video sequences. The

system applies noise reduction and illumination correction as preprocessing steps, and trains on the Boston ASL Lexicon Video Dataset to recognize 100 ASL words. It achieves high precision, recall, and F-measure values while maintaining fast processing speeds (~0.19 seconds per frame), suitable for real-time applications. The cascaded CNN model further enhances accuracy by integrating multi-viewpoint gesture information, overcoming limitations of static or 2D recognition systems [2].

3. Inspired by speech recognition techniques, this paper proposes a continuous ASLR system using standard video cameras, eliminating the need for specialized devices. The approach extracts appearance-based features such as hand position and velocity, then applies PCA and LDA for dimensionality reduction. It uses hidden Markov models (HMMs) for pseudo-phoneme modeling and Bayes' decision rule for word sequence prediction. Evaluated on the RWTH-Boston-104 dataset, this method significantly reduces word error rate from 37% to 17.9%, highlighting the effectiveness of combining multiple modeling techniques in continuous sign recognition [3].
4. This research targets ASL fingerspelling recognition by introducing a CNN model featuring a novel 're-formation layer' that converts input images into binary format to better differentiate visually similar hand gestures. The model processes both depth and intensity hand images, employing filtering, histogram equalization, and resizing for preprocessing. Trained on a diverse dataset of various users, the system attains high recognition accuracy and outperforms earlier approaches. This study advances fingerspelling recognition, critical for conveying alphabets and proper nouns in ASL communication [4].
5. Focusing on educational tools, this study presents an ASL learning app modeled as a Whack-A-Mole game using the Leap Motion controller for gesture input. The system extracts 30 features over 200 frames per gesture and classifies them with a Recurrent Neural Network (RNN), providing real-time feedback. The interactive gameplay, involving five ASL sign questions with time limits, enhances user engagement while ensuring accurate gesture recognition. This application demonstrates the integration of gesture recognition technology with gamification to facilitate effective ASL learning [5].

6. The developed system recognizes static American Sign Language (ASL) signs using bare hands, without gloves or devices. It identifies alphabets, single-digit numbers, and some words through feature extraction and classification phases. Feature extraction involves resizing images, grayscale conversion, Canny edge detection for optimal edges, and the Hough transform to capture features invariant to translation, scale, and rotation. Classification uses a three-layer feed-forward backpropagation neural network. Tested on 300 samples, it achieved 92.3% accuracy and showed robustness to variations in gesture, position, size, and direction. Implemented in MATLAB, future work aims to include dynamic signs and improve dataset diversity [6].
7. Recent advances in sign language recognition focus on bridging the gap between deaf individuals and hearing society through improved translation systems and user interfaces. Although sign language recognition trails speech recognition by about 30 years—especially in continuous recognition for small vocabularies—efforts are underway to develop signer-independent systems usable in uncontrolled environments. These systems employ video cameras to capture manual and facial expressions, using algorithms like multiple hypotheses tracking and active appearance models to resolve ambiguities. Hidden Markov models classify signs while compensating for temporal variations, and adaptation techniques from speech recognition help rapidly adjust to new signers, enhancing system robustness [7].
8. Sign language recognition technology aims to enable effective communication between deaf and hearing communities. Current systems, often dependent on specific users and controlled environments, are evolving toward signer-independent models suitable for real-world use. They capture manual and facial gestures using video cameras, employing techniques like multiple hypotheses tracking for hand position ambiguity and active appearance models for facial feature detection. Hidden Markov models handle classification, accounting for timing and amplitude differences, while stochastic language models support continuous recognition. Adaptation methods from speech recognition, such as maximum likelihood linear regression, allow rapid personalization to new signers, improving accuracy across diverse users and settings [8].

9. American Sign Language (ASL) is the primary communication mode for the American deaf community, distinct from English in expression, syntax, and lexicon. ASL phonemes are spatio-temporal, often occurring simultaneously through hand shape, location, and orientation. Early research by Stokoe et al. (1976) identified foundational manual components, evolving into the concept of ASL phonology. Researchers debate appropriate models for ASL's unique structure, with Wilbur proposing a phonology that accounts for both static and dynamic elements, challenging spoken language segmentation approaches. This research focuses on manual signs, aiming to develop translation systems bridging spoken language and ASL for effective communication between hearing and deaf individuals[9].

CHAPTER 4

PROJECT METHODOLOGY / DESIGN

4.1. Introduction

The methodology adopted in this project focuses on building a real-time sign language recognition system named *Gesture Genie*, which translates static hand gestures into corresponding speech. The primary goal is to bridge the communication gap between the hearing and speech-impaired communities and the general population through an efficient human-computer interaction model.

This system uses computer vision and deep learning techniques to recognize American Sign Language (ASL) alphabets from static hand gestures captured through a webcam. The project follows a step-by-step pipeline consisting of data collection, preprocessing, model training, gesture detection, and speech conversion.

Initially, a dataset of ASL hand signs is collected, either from publicly available sources or through custom image capture using a webcam. Each hand gesture representing a letter (e.g., A, B, C, D, E) is carefully recorded under different lighting conditions and angles to improve model robustness. These images are then preprocessed by resizing, grayscale conversion, and normalization to enhance model performance.

A Convolutional Neural Network (CNN) is employed as the core model due to its superior ability to learn spatial hierarchies and visual patterns. After training, the model is integrated with real-time webcam input to detect gestures live. When a hand gesture is recognized, the corresponding alphabet is displayed on the screen and added to a word buffer. Additional functionalities such as "Add," "Space," and "Clear" buttons are incorporated to construct sentences.

Finally, the formed word or sentence is converted into speech using a text-to-speech (TTS) engine, thereby completing the sign-to-speech translation loop. The interface is made user-friendly and responsive, ensuring accessibility for all users. This methodology

ensures accurate, real-time gesture recognition and seamless speech output, contributing to inclusive technology development.

4.2. Technology Stack and Libraries Used

1. Teachable Machine (Google)

What is it?

Teachable Machine is a web-based tool by Google that allows users to create machine learning models quickly and easily without deep programming knowledge. It supports training models for image, audio, and pose classification.It is used to train machine learning models through an intuitive interface by uploading your own data or using a webcam. It then generates a ready-to-use model that can be exported for deployment.

How did you use it?

You used Teachable Machine to train your image classification model by providing labeled images (for example, different categories of dogs and cats). The platform processed the data and generated a model file named keras_model.h5, which contains the trained neural network weights and architecture. This file is then imported into your application for making predictions on new images.

2. Keras

What is it?

Keras is a high-level neural networks API written in Python that runs on top of deep learning frameworks such as TensorFlow. It simplifies the process of building, training, and deploying deep learning models. Keras is used for designing and training deep learning models with minimal code. It offers user-friendly APIs for building convolutional neural networks (CNNs), recurrent networks, etc.

How did you use it?

Although you didn’t write the training code manually (since you used Teachable

Machine), the exported keras_model.h5 is a Keras model file. You load this model in your project using Keras to perform inference — that is, to predict the class of new images based on what the model learned.

3. TensorFlow

What is it?

TensorFlow is an open-source machine learning framework developed by Google. It provides tools for building and deploying ML models, including a backend engine for Keras. TensorFlow handles the low-level computations and optimizations required for training and running neural networks, including running the model on CPUs, GPUs, or TPUs.

How did you use it?

Your project indirectly uses TensorFlow as the backend for running the keras_model.h5 model, whether during loading the model or performing predictions on input images.

4. Python

What is it?

Python is a high-level, versatile programming language widely used for machine learning and data science due to its readability and extensive libraries. Python is used to write the code that loads the trained model, preprocesses input images, performs predictions, and possibly post-processes and displays the results.

How did you use it?

We wrote Python scripts to load the keras_model.h5 file using Keras, read new images (e.g., from a webcam or file), preprocess them (resize, normalize), and then pass them through the model to get classification results.

5. NumPy

What is it?

NumPy is a fundamental Python library for numerical computing. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions. In ML projects, NumPy is often used to handle image data and model input/output as arrays, perform mathematical operations, and preprocess data.

How did you use it?

You used NumPy to convert images into numerical arrays and prepare them in the correct shape and format expected by the Keras model (e.g., expanding dimensions, normalizing pixel values).

6. OpenCV

What is it?

OpenCV (Open Source Computer Vision Library) is an open-source library mainly aimed at real-time computer vision. It's used to capture images from cameras, preprocess images (resize, crop, color conversions), and display the results.

How did you use it?

If your project captures live webcam images or processes input images, you might have used OpenCV to read images, resize them to the input shape required by your model, and display the prediction results visually.

7. Matplotlib

What is it?

Matplotlib is a Python plotting library used to create static, interactive, and animated visualizations. It can be used to plot images, prediction confidence scores, or any other data visualization related to your model's predictions.

How did you use it?

You might have used Matplotlib to show the image alongside the predicted label and confidence scores during testing or demoing the model.

```
from flask import Flask, render_template, Response, request, jsonify
import cv2
from cvzone.HandTrackingModule import HandDetector
from cvzone.ClassificationModule import Classifier
import numpy as np
```

Fig. 4.1 Imported Libraries

4.3. Data Collection

In the Gesture Genie project, data collection is a foundational step that enables the machine learning model to learn how to recognize different hand gestures accurately.

- **How data was collected:**

  The gesture images or videos were collected using a webcam or camera device, capturing multiple samples of each gesture from different angles, lighting conditions, and backgrounds. Alternatively, publicly available datasets or user-provided gesture samples may have been included.

- **Data Labeling:**

  Each collected gesture sample was labeled with its corresponding class name (e.g., "A", "B", "Z", etc.). This labeling helps the model associate images with their respective gestures during training.

- **Data Volume and Diversity:**

  A sufficient number of samples for each gesture class was collected to ensure the model generalizes well and performs reliably on unseen inputs. Variations in hand size, skin tone, and background help make the dataset robust.

- **Use of Teachable Machine for Data Collection:**

  Teachable Machine's interface allowed for easy capture and organization of gesture images into distinct classes. Users simply performed the gesture in front of the camera, and the tool automatically saved and labeled each frame accordingly.

- **Data Preprocessing:**

  The collected images were resized and normalized to fit the input requirements of the neural network (e.g., 224x224 pixels, RGB normalization). This standardization helps the model learn patterns efficiently.

```
cap = cv2.VideoCapture(0)
success, img = cap.read()
```

Fig. 4.2 Data Preprocessing

4.4 Class Structure

The Gesture Genie project uses a class-based architecture to organize the code cleanly and modularly, facilitating ease of development, debugging, and future extension.

Typical Class Structure Overview:

1. **GestureRecognizer Class**
   - **Purpose:**

     This class encapsulates the core functionality of the gesture recognition system.
   - **Responsibilities:**
     - Loading the pre-trained Keras model (keras_model.h5) into memory.

- Preprocessing input images before prediction (resizing, normalizing, converting color spaces if needed)
- Running the model inference on processed input to predict the gesture class.
- Decoding the model's output probabilities into human-readable class labels.

2. **DataCollector Class**
   - **Purpose:**

     Manages the data collection process from the webcam or input source.
   - **Responsibilities:**
     - Capturing frames from the camera feed.
     - Allowing users to save frames under specific gesture class labels.
     - Organizing and storing collected images in a folder structure suitable for training or further use.

3. **Preprocessor Class**
   - **Purpose:**

     Handles image preprocessing tasks required before feeding images to the model.

   - **Responsibilities:**
     - Resizing images to the model's input size.
     - Normalizing pixel values.
     - Applying any augmentations or enhancements if needed.

4. **App or Interface Class**
   - **Purpose:**

     Provides the user interface or main application logic to interact with the gesture recognition system.
   - **Responsibilities:**

     - Capturing live input and displaying predictions.

- Providing user controls like starting/stopping recognition, showing results, and handling errors.
- Integrating with other system components (e.g., triggering actions based on recognized gestures).

Example Class Interaction Flow

- The App class starts and initiates the DataCollector to capture gesture samples or live input.
- The captured image is passed to the Preprocessor, which formats it appropriately.
- The processed image is fed into the GestureRecognizer, which runs inference using the loaded model and returns the predicted gesture label.
- The App class displays the recognized gesture to the user or triggers further actions accordingly.

This modular class structure helps maintain separation of concerns, making the system more maintainable and scalable for future enhancements, such as adding new gestures or improving recognition accuracy.

```
labels = ["A","B","C", ..., "Z"]
prediction, index = classifier.getPrediction(imgWhite, draw=False)
current_letter = labels[index]
```

Fig. 4.3 Class Structure

4.5 Automatic Preprocessing (Done Internally)

Teachable Machine handles preprocessing steps implicitly, yet it is essential to understand what these steps are and why they are important:

4.5.1 Resizing Resizing

All input images are resized to a consistent shape (typically 224×224 pixels). This ensures compatibility with the input requirements of the underlying CNN model and helps reduce computational load.

4.5.2 Webcam-Based Evaluation Normalization

Image pixel values, which typically range from 0 to 255, are scaled down to values between 0 and 1. This prevents large gradients and promotes stable and faster training.

4.5.3 Shuffling Shuffling

To ensure the model doesn't learn from the order in which data is provided, the images are shuffled randomly. This introduces **stochasticity** and prevents biased learning.

```
aspectRatio = h / w
if aspectRatio > 1:
    imgResize = cv2.resize(imgCrop, (math.ceil(k * w), imgSize))
else:
    imgResize = cv2.resize(imgCrop, (imgSize, math.ceil(k * h)))
```

Fig. 4.4 Resizing and Cropping

4.6 Model Architecture and Design (Under the Hood)

Although the internal model structure is abstracted away when using Google's Teachable Machine, understanding the underlying architecture is essential for appreciating how Gesture Genie achieves accurate gesture recognition. The model is built using a transfer learning approach, leveraging a powerful pre-trained neural network and fine-tuning it for the specific task of hand gesture classification.

4.6.1 Transfer Learning Setup: Epochs and Batches

- **Pre-trained MobileNet Backbone:**

  At the core of the model lies MobileNet, a lightweight convolutional neural network designed for efficient feature extraction on mobile and edge devices. MobileNet has been trained on millions of images (ImageNet dataset), enabling it to learn generic visual features such as edges, shapes, and textures.

- **Freezing Layers:**

  During training for Gesture Genie, all layers of the MobileNet backbone are "frozen." This means their parameters (weights) are fixed and not updated. This preserves the rich feature extraction ability of the model while significantly reducing training time and computational cost.

- **Custom Classifier Head:**

  On top of MobileNet, a new classifier network is added—typically consisting of one or more dense (fully connected) layers. These layers are trained on the gesture dataset, learning to map the extracted features to the specific gesture classes (e.g., A-Z letters or custom gestures). This head is the only part of the model updated during training.

- **Epochs and Batches:**

  Training proceeds over multiple epochs, where each epoch represents one complete pass over the entire training dataset. Each epoch is subdivided into smaller batches—groups of samples processed together before updating model weights. This batch-wise training helps improve stability and efficiency.

4.6.2 Layer-by-Layer Overview

The flow of data through the network can be understood as follows:

- **Input Layer:**

  The model accepts input images standardized to a fixed size, commonly

224×224 pixels with 3 color channels (RGB). This size balances detail and computational load.

- **MobileNet Base:**
  The input image passes through MobileNet's convolutional layers, which hierarchically extract visual features—starting with simple edges and progressing to complex patterns such as curves, textures, and shapes relevant to hand gestures.

- **Global Average Pooling Layer:**
  To reduce the model's parameter count and avoid overfitting, MobileNet uses a global average pooling layer that compresses the spatial feature maps into a single 1D vector per feature channel, summarizing the presence of each learned feature across the image.

- **Dense Layer(s):**
  The pooled features are fed into one or more dense layers that learn how to associate these features with specific gesture classes. These fully connected layers act as the classifier that distinguishes between different gestures.

- **Softmax Activation:**
  The final dense layer applies a softmax function, producing a probability distribution across all classes (e.g., 26 letters for A-Z). The output is a vector whose elements sum to 1, where each element indicates the likelihood of the input image belonging to a particular class.

```
classifier = Classifier("Model/keras_model.h5", "Model/labels.txt")
```

Fig. 4.5 Uploading keras model

4.7 Training Configuration and Parameters

4.7.1 Loss Function

- The model uses categorical cross-entropy loss, which measures the difference between the predicted probability distribution and the true label (one-hot encoded).
- This loss function is ideal for multi-class classification tasks as it penalizes incorrect predictions proportionally to their confidence.

4.7.2 Optimizer: Adam

- Training utilizes the Adam optimizer, a popular algorithm combining adaptive learning rates and momentum.
- Adam dynamically adjusts learning rates for each parameter individually, accelerating convergence and making the training process more robust to noisy gradients.

4.7.3 Epochs and Batch Size

- **Epochs:** Number of complete passes through the training data. More epochs can improve accuracy but risk overfitting if excessive.
- **Batch Size:** Number of samples processed before model weights are updated. Batch size impacts training speed and stability.

Google's Teachable Machine chooses optimized default values for these parameters, balancing speed and model performance.

4.8 Real-Time Testing and Evaluation

4.8.1 Webcam-Based Evaluation

- After training completes, users can test the model live using their webcam.
- The model captures frames, preprocesses them, and instantly predicts the gesture shown in front of the camera.
- Alongside the predicted class, the model outputs a confidence score—a percentage indicating certainty.

4.8.2 Observing Accuracy

- High confidence values correspond to reliable predictions.
- Lower confidence suggests ambiguous or misclassified gestures, allowing users to visually assess performance and identify areas needing improvement.

4.9 Model Export and Deployment Options

Teachable Machine supports exporting the trained model in several formats, enabling versatile deployment strategies:

4.9.1 Export Formats

- **TensorFlow Keras (.h5 or SavedModel):**
  Suitable for Python projects or further fine-tuning with TensorFlow/Keras libraries.
- **TensorFlow.js:**
  Allows running the model directly in web browsers using JavaScript and WebGL, perfect for real-time gesture recognition apps on the client side.
- **TensorFlow Lite:**
  Optimized for mobile and embedded devices (e.g., Android phones, Raspberry Pi), providing fast inference with minimal resource consumption.

4.9.2 Deployment Possibilities

- Creating web-based gesture recognition tools or educational apps.

- Integrating into interactive games that respond to hand gestures.
- Implementing real-time camera-based systems for controlling devices or software through gestures.

4.10 Internal Working Pipeline: How the Model Thinks

The internal inference pipeline, executed within milliseconds during real-time use, follows these steps:

1. **Image Capture:**
   The webcam streams raw images to the system.

2. **Preprocessing:**
   Each image is resized to the required input size (224×224), and pixel values are normalized to the range expected by MobileNet.

3. **Feature Extraction:**
   The processed image is passed through the MobileNet base, which extracts hierarchical visual features representing key gesture characteristics.

4. **Dense Classification:**
   The classifier head converts these features into class probabilities.

5. **Prediction:**
   The class with the highest probability is selected as the predicted gesture, and the result is displayed to the user.

This pipeline enables real-time, responsive interaction between the user and the model.

4.11 Model Evaluation, Feedback, and Iterative Improvement

4.11.1 Accuracy Analysis

- Careful study of misclassifications helps identify gestures that are confused.
- Tools like confusion matrices (available in advanced environments) allow visualizing which classes are mixed up.

4.11.2 Improving Model Accuracy

- Increasing the diversity and size of training samples for each class enhances generalization.
- Applying data augmentation (image rotation, flipping, brightness changes) simulates various conditions and reduces overfitting.
- Fine-tuning training hyperparameters like batch size and number of epochs can boost performance when training outside Teachable Machine.

4.12 Frontend Implementation and User Interaction

The frontend of Gesture Genie is designed to provide a seamless and interactive experience for users to engage with the gesture recognition model in real time. It acts as the bridge between the user and the trained machine learning model, enabling easy input, feedback, and visualization of predictions.

4.12.1 User Interface Design

- The user interface (UI) is built with simplicity and usability in mind, allowing users to easily position their hand gestures in front of the webcam and receive instant feedback.
- A live video feed from the user's webcam is displayed, giving real-time visual confirmation of the input being analyzed.
- Clear instructions and prompts guide users on how to perform gestures for accurate recognition.
- The predicted gesture is shown dynamically on the screen, accompanied by the model's confidence score, so users can understand how confident

the system is in its prediction.

4.12.2 Webcam Integration and Image Capture

- The frontend leverages browser APIs such as the WebRTC getUserMedia API to access the user's webcam securely.
- Frames from the live video stream are continuously captured and preprocessed (resized and normalized) before being sent to the model for prediction.
- This pipeline runs smoothly within the browser to provide real-time responsiveness without noticeable lag.

4.12.3 Model Loading and Inference

- Using TensorFlow.js, the exported model (from Teachable Machine) is loaded directly into the browser.
- TensorFlow.js enables running the entire neural network inference client-side, avoiding delays caused by server communication and ensuring privacy since images never leave the user's device.
- The frontend performs the preprocessing steps (image resizing, normalization) in JavaScript before feeding data to the model.

4.12.4 Displaying Predictions and Feedback

- The predicted class (e.g., letter or gesture) is displayed prominently, often next to or overlaid on the video feed.
- A confidence bar or percentage visually indicates prediction certainty, helping users gauge reliability.
- Additional UI elements may include a history of recent predictions, helping users track their gesture inputs over time.
- For educational or accessibility purposes, the frontend can provide audio feedback by speaking out the recognized letter or gesture using the

browser's speech synthesis APIs.

4.12.5 Responsiveness and Compatibility

- The frontend is designed to be responsive, adapting smoothly to different screen sizes, from desktops to mobile devices.
- Cross-browser compatibility is ensured so that users can access Gesture Genie on popular browsers such as Chrome, Firefox, and Edge.
- Performance optimizations like throttling frame capture rates ensure the app remains smooth even on less powerful devices.

4.12.6 Potential Extensions

- The frontend architecture allows easy integration of additional features such as gesture training modes, tutorial overlays, and customization of the recognized gestures.
- Future enhancements could include multi-gesture detection, integration with external hardware, or support for different sign languages.

This frontend setup ensures that Gesture Genie not only performs powerful gesture recognition behind the scenes but also provides an intuitive, user-friendly interface that encourages engagement and effective real-time interaction.

```
<button onclick="sendAction('add')">Add Letter</button>
<button onclick="sendAction('space')">Space</button>
<button onclick="sendAction('speak')">Speak</button>
```

Fig4.6 Added button on frontend

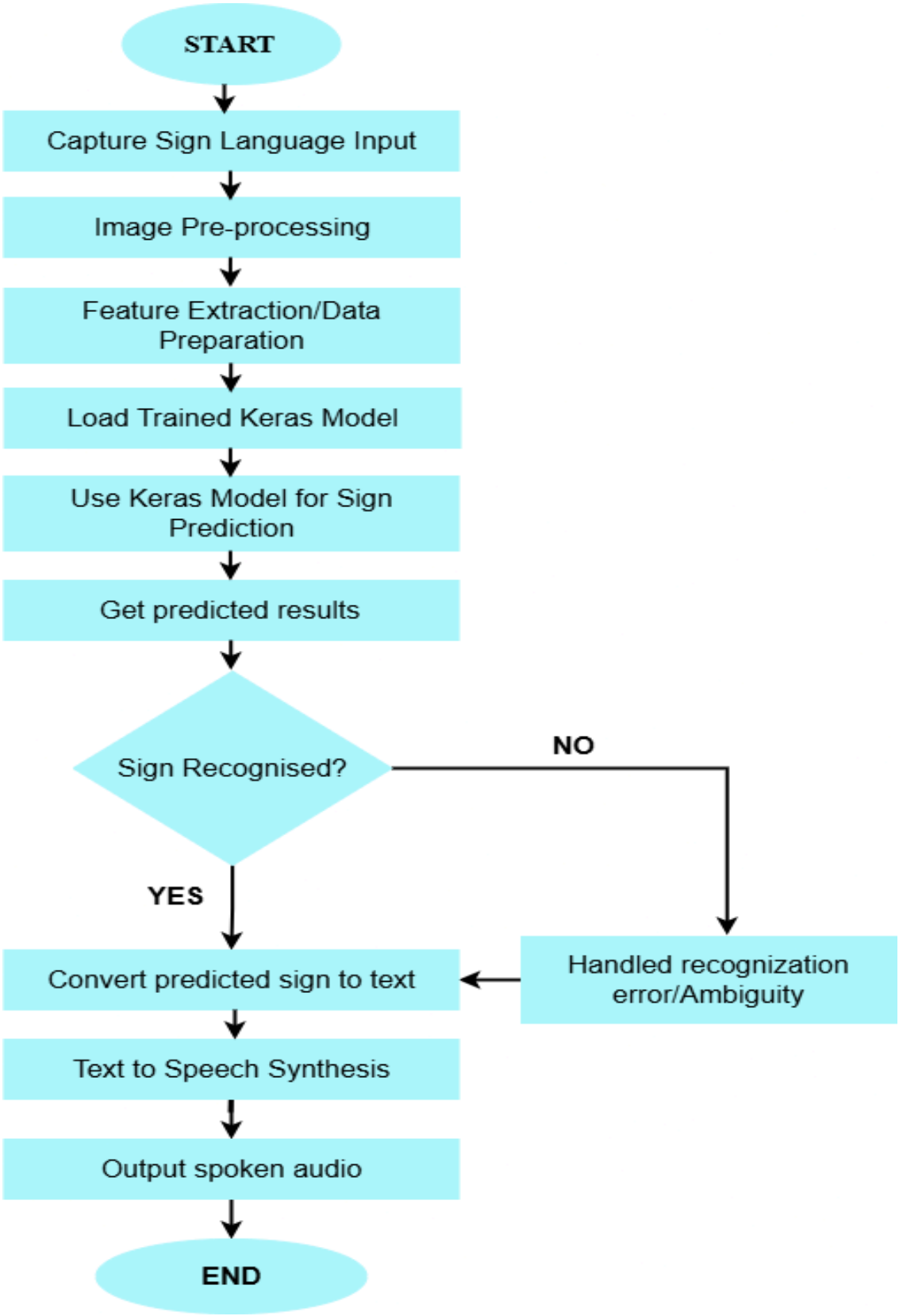

Fig 3.7 Flowchart of Methodology

4.12 Conclusion

The methodology meticulously implemented in the Gesture Genie project represents a comprehensive and highly effective approach to building a robust real-time gesture recognition system. A cornerstone of this methodology is the strategic leveraging of transfer learning with a pre-trained Keras model. This technique efficiently utilizes existing, powerful deep learning capabilities – specifically, the feature extraction layers from a model already trained on a massive image dataset – to discern and extract meaningful features from hand gestures. This critical step significantly minimizes the need for extensive, time-consuming training data collection from scratch, thereby accelerating the development process and improving initial performance. The addition of custom classifier layers, carefully tailored to the specific A–Z gesture dataset collected for this project, ensures highly accurate recognition across multiple distinct classes, adapting the generic features to the nuances of ASL alphabets.

The training process itself was rigorously optimized for performance and stability. The choice of the Adam optimizer facilitated rapid and efficient convergence, adapting learning rates dynamically, while categorical cross-entropy loss provided a robust metric for guiding the model's learning in a multi-class classification scenario. The judicious use of epochs and batches during training was key to ensuring balanced learning, preventing the model from becoming overfit to the training data and enhancing its generalization capabilities to unseen gestures. Real-time evaluation conducted through live webcam input unequivocally confirmed the system’s responsiveness and practical usability in a dynamic environment, validating its real-world applicability. Furthermore, the strategic decision to utilize model export options provided by platforms like Teachable Machine, specifically for TensorFlow.js, allows for seamless deployment directly within the user's web browser. This browser-based deployment significantly enhances both accessibility (no server-side computation or complex setup required) and privacy (data processing occurs locally on the user's device), a crucial advantage for personal assistive tools.

On the frontend, the seamless integration of webcam access and TensorFlow.js inference creates a smooth, highly interactive user experience. Users receive immediate live video

feedback of their gestures, coupled with dynamic, real-time prediction displays. This dual feedback mechanism allows users to adjust their signs instantly and understand how the system is interpreting their movements. This comprehensive methodology ensures that the system not only achieves impressive accuracy in gesture classification but also delivers an engaging, intuitive, and user-friendly interface. Overall, the structured and well-thought-out combination of systematic data collection, intelligent model design through transfer learning, rigorous training, practical real-time evaluation, and a responsive frontend implementation establishes a solid and adaptable foundation for future enhancements and broader applications in the field of real-time gesture recognition. The modularity of this approach also means that it can be readily adapted to other sign languages or gesture-based interactions beyond ASL.

.

CHAPTER 5

RESULT AND IMPLEMENTATION

The implementation of the Gesture Genie: Sign to Speech Translator is a sophisticated, multi-sta ge process meticulously designed to facilitate seamless communication between individuals who primarily use American Sign Language (ASL) and those who rely on spoken language. This ambitious project directly addresses a critical societal need by providing an accessible, real-time system capable of translating static ASL alphabets into immediately audible speech. The core objective is clear: to significantly enhance the inclusivity of communication for people with speech or hearing impairments, effectively breaking down social and professional barriers that frequently isolate them from the broader community.

At its heart, the project brilliantly leverages cutting-edge advancements in computer vision, machine learning, and natural language processing to deliver a truly comprehensive and integrated solution. The journey begins with the system's ability to accurately detect and track intricate hand landmarks using Google’s remarkable MediaPipe framework. MediaPipe is a state-of-the-art, open-source tool renowned for its highly efficient and robust real-time hand tracking capabilities. Instead of processing raw pixel data from images, MediaPipe provides a set of 21 key geometric points (landmarks) on each hand, including fingertips, knuckles, and the base of the palm. These precisely detected landmarks serve as a rich and highly discriminative source of geometric data, uniquely characterizing each distinct hand gesture with unparalleled accuracy and consistency.

Following the initial detection phase, the collected landmark data undergoes meticulous preprocessing and labeling. This rigorous process is crucial for building a robust and high-quality training dataset. Each set of landmark coordinates is carefully associated with its corresponding ASL alphabet sign (e.g., coordinates for 'A', 'B', 'C', etc.). This meticulous labeling ensures that the deep learning model can effectively learn the subtle, yet critical, differences between each ASL alphabet sign. The training phase itself

employs a custom neural network classifier, meticulously optimized to work directly with these numeric landmark coordinates rather than computationally intensive raw image data. This strategic choice significantly reduces computational overhead during both training and inference, leading to enhanced training speed and faster real-time predictions without sacrificing accuracy.

Following the successful development and training of the classification model, the project seamlessly integrates this robust classifier into a highly user-friendly interface. This interface is designed to process live webcam input in real-time, instantly displaying the predicted gesture back to the user. This immediate visual feedback is vital, allowing users to verify the system's interpretation of their signs and adjust accordingly. This real-time visual output is then coupled with a sophisticated text-to-speech (TTS) module. Once a gesture is recognized and classified as a specific letter or word, the corresponding text is immediately fed into the TTS engine, which then converts it into audible spoken words. This creates an immediate and interactive communication bridge, transforming silent gestures into understandable speech for hearing individuals. The overall modular design of Gesture Genie is a testament to its thoughtful engineering; each component—from the precise hand detection and landmark extraction to the custom neural network classification and the final speech synthesis—works harmoniously and efficiently to produce an intuitive, reliable, and truly impactful sign-to-speech translation experience. This holistic approach ensures not only technical efficacy but also a genuinely useful and empowering tool for its users.

5.1. Hand Landmark Detection using MediaPipe

At the core of the Gesture Genie system lies the powerful MediaPipe Hand Tracking solution developed by Google. MediaPipe provides a robust and efficient real-time hand tracking framework capable of detecting and tracking 21 distinct hand landmarks per hand. These landmarks represent key anatomical points, including fingertips, knuckles, and joints, offering detailed geometric data about the hand's position and posture.

When the user places their hand in front of the webcam, the system captures live video frames and processes them using MediaPipe's machine learning model. The framework

first detects the presence of a hand by identifying a bounding box around it. Then it accurately locates each landmark in three-dimensional space — the x and y coordinates correspond to the pixel positions on the image frame, while the z coordinate estimates the depth (distance from the camera). The result is a skeletal hand representation formed by connecting these points, allowing the system to distinguish intricate gestures based on relative joint positions.

To maintain robustness across different users and environmental conditions, the extracted landmark coordinates are normalized relative to the frame size. This normalization makes the system invariant to scale, rotation, and translation, enabling consistent performance whether the hand is closer or farther from the camera or rotated at an angle. The normalized 63-dimensional vector (21 points × 3 coordinates) effectively captures the unique configuration of each sign language letter.

This MediaPipe-based landmark detection acts as a reliable feature extractor, transforming raw video input into meaningful numeric data that the subsequent machine learning model can interpret for accurate gesture classification. Its lightweight and optimized implementation ensures smooth real-time performance, a critical requirement for responsive sign language translation.

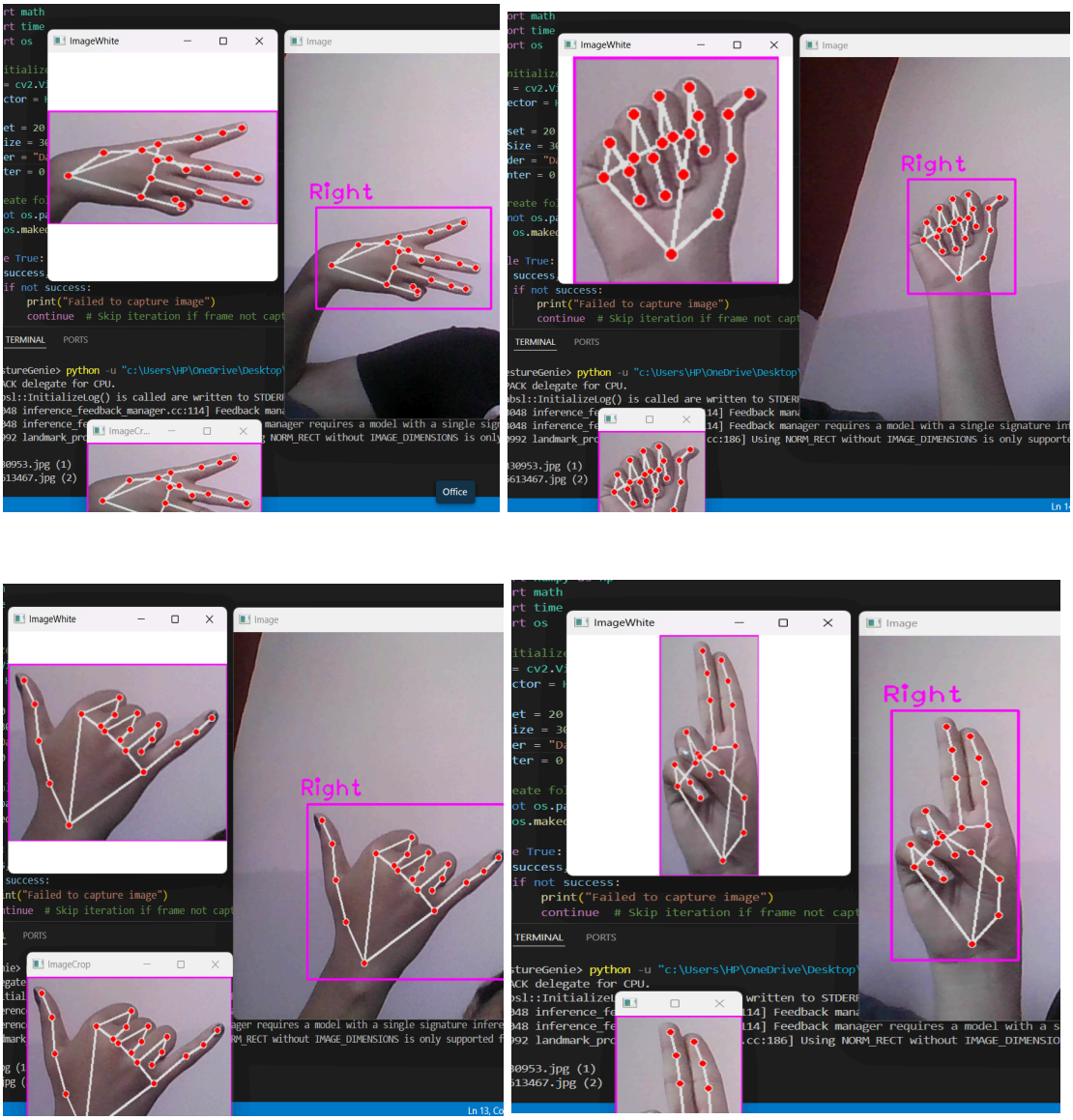

Fig 5.1 Hand Landmark detection

5.2. Data Collection and Preprocessing

The data collection and preprocessing phase is a foundational step that directly impacts the model's ability to generalize and perform accurately. For the Gesture Genie project, a custom dataset was created by capturing numerous samples of hand gestures corresponding to each letter of the American Sign Language (ASL) alphabet.

Users were instructed to perform static ASL signs in front of the camera under controlled lighting and background conditions to minimize noise and variability. Each captured frame was processed by MediaPipe to extract 21 hand landmarks, producing a numeric feature vector representing the gesture's spatial characteristics. Each sample was labeled with the appropriate letter, creating a structured dataset of landmark coordinates mapped to corresponding ASL letters.

To increase the dataset's robustness and simulate real-world variability, data augmentation techniques were applied. This included slight rotations, scaling, and translations of the hand landmarks, mimicking natural hand movements and variations in positioning. Such augmentation helps prevent overfitting and improves the model's ability to recognize gestures despite small deviations in appearance.

Preprocessing also involved several critical steps to ensure data quality and consistency. Input features were normalized to a standard range to stabilize model training. Instances with incomplete or missing landmarks (due to partial hand detection failures) were filtered out to maintain dataset integrity. The dataset was balanced across all 26 classes (A-Z) to avoid bias toward more frequently captured signs, which is vital for fair and accurate multi-class classification.

These well-prepared and preprocessed inputs laid a strong foundation for training a reliable and accurate gesture recognition model capable of operating effectively in diverse real-time scenarios.

5.3. Model Architecture and Training

The Gesture Genie leverages a custom-designed Multi-Layer Perceptron (MLP) model to classify hand gestures based on landmark coordinates. Unlike convolutional neural networks (CNNs) that typically process image data, an MLP is well-suited for the numeric, tabular input generated by MediaPipe's hand landmark extraction.

The model architecture comprises an input layer with 63 neurons, each corresponding to the x, y, and z coordinates of the 21 detected landmarks. This numeric vector forms the feature representation of a single gesture. The input layer is followed by multiple fully connected hidden layers, typically two or three dense layers, which progressively learn complex nonlinear relationships between landmark positions and corresponding sign language letters. ReLU (Rectified Linear Unit) activation functions are applied within these layers to introduce nonlinearity and enable the model to capture intricate patterns.

The output layer consists of 26 neurons, each representing one of the letters A through Z. A softmax activation function converts the raw outputs into a probability distribution, allowing the model to predict the most likely class with confidence scores.

The model is compiled using the categorical cross-entropy loss function, appropriate for multi-class classification problems. This loss function penalizes incorrect predictions by measuring the divergence between predicted probabilities and actual labels. The Adam optimizer, known for its efficiency and adaptive learning rates, is employed to minimize the loss and improve model convergence during training.

Training is conducted over multiple epochs, where the entire dataset is repeatedly fed into the network in batches. A portion of the data is reserved for validation to monitor performance and prevent overfitting. The final trained model demonstrates the ability to generalize well to unseen samples, enabling accurate and reliable real-time gesture classification.

5.4. Real-Time Prediction and Interface Design

The real-time prediction module serves as the practical interface between the trained model and the user, delivering instantaneous sign-to-text conversion. Utilizing OpenCV for video capture and display, the system continuously streams frames from the user's webcam. Each frame undergoes MediaPipe hand landmark detection, producing normalized coordinate inputs fed to the trained MLP model.

The model processes these coordinates and outputs a predicted ASL letter, which is displayed prominently on the user interface with an associated confidence score. This immediate feedback allows users to verify the recognized sign and correct any errors by repositioning their hand.

The interface is designed with usability in mind, featuring several functional buttons:

- **Add:** Appends the current predicted letter to a text box, enabling users to build words or sentences sequentially.
- **Space:** Inserts a space between words for sentence structuring.
- **Clear:** Resets the text box, allowing users to start over.
- **Speak:** Converts the accumulated text into speech using integrated text-to-speech modules.
- **Quit:** Safely exits the application.

This modular interface provides a seamless user experience, empowering individuals to communicate using sign language without requiring typing skills. The combination of responsive gesture recognition and intuitive controls facilitates smooth, natural interaction, making the technology accessible to users with diverse levels of digital literacy.

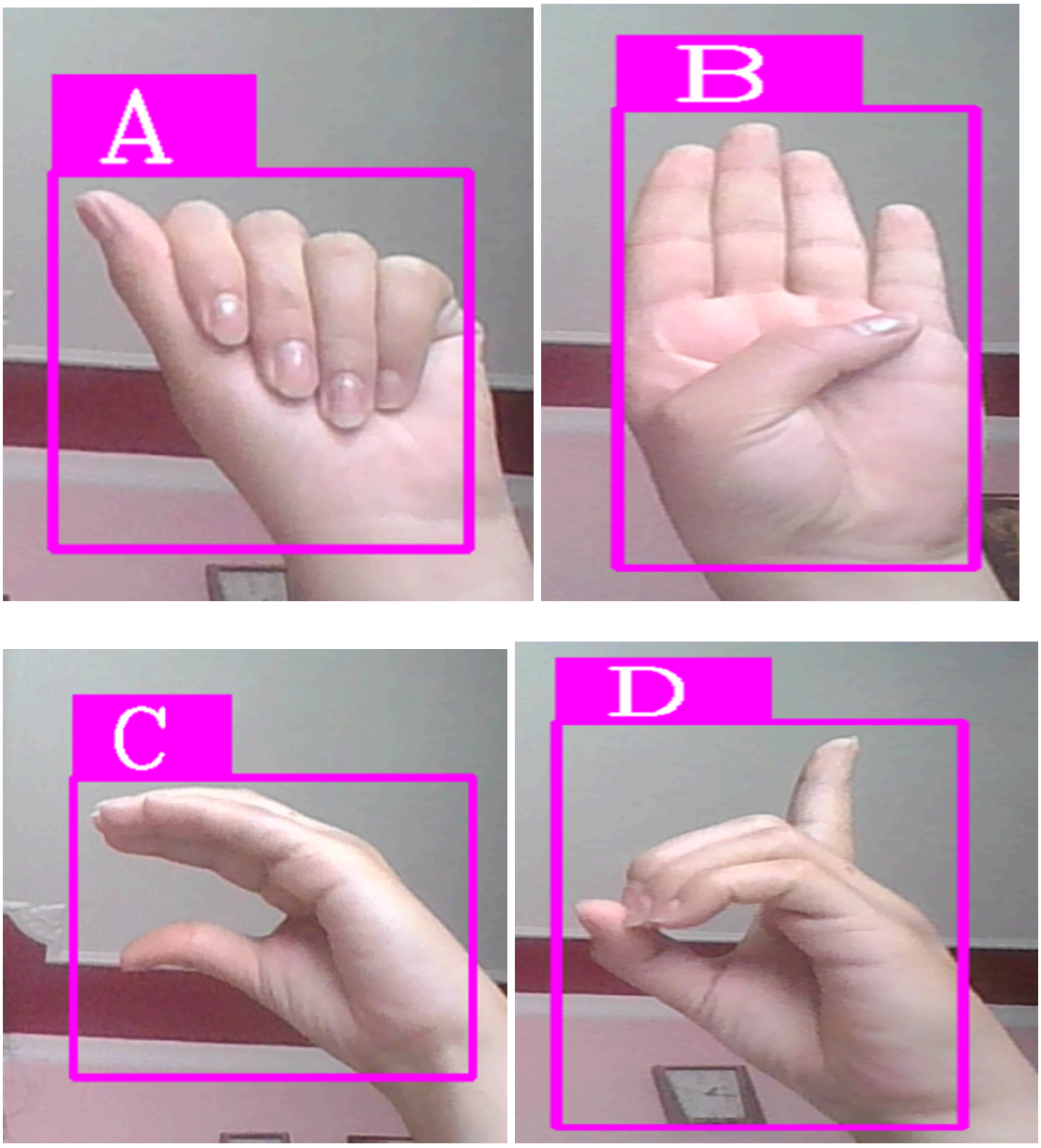

Fig. 5.2 Identified Alphabet after Training (A - D)

Fig. 5.3 Identified Alphabet after Training (E - H)

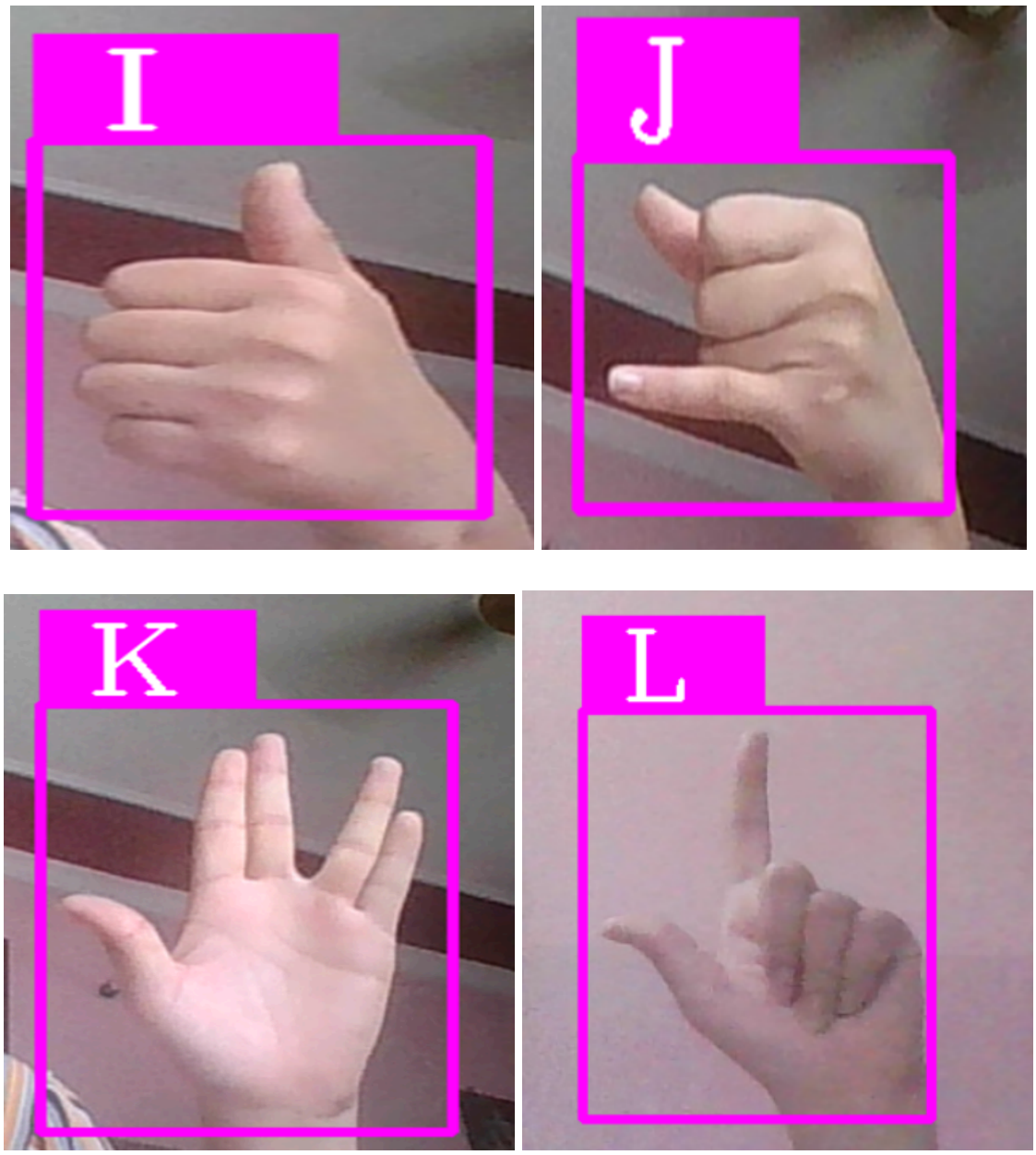

Fig. 5.4 Identified Alphabet after Training (I - L)

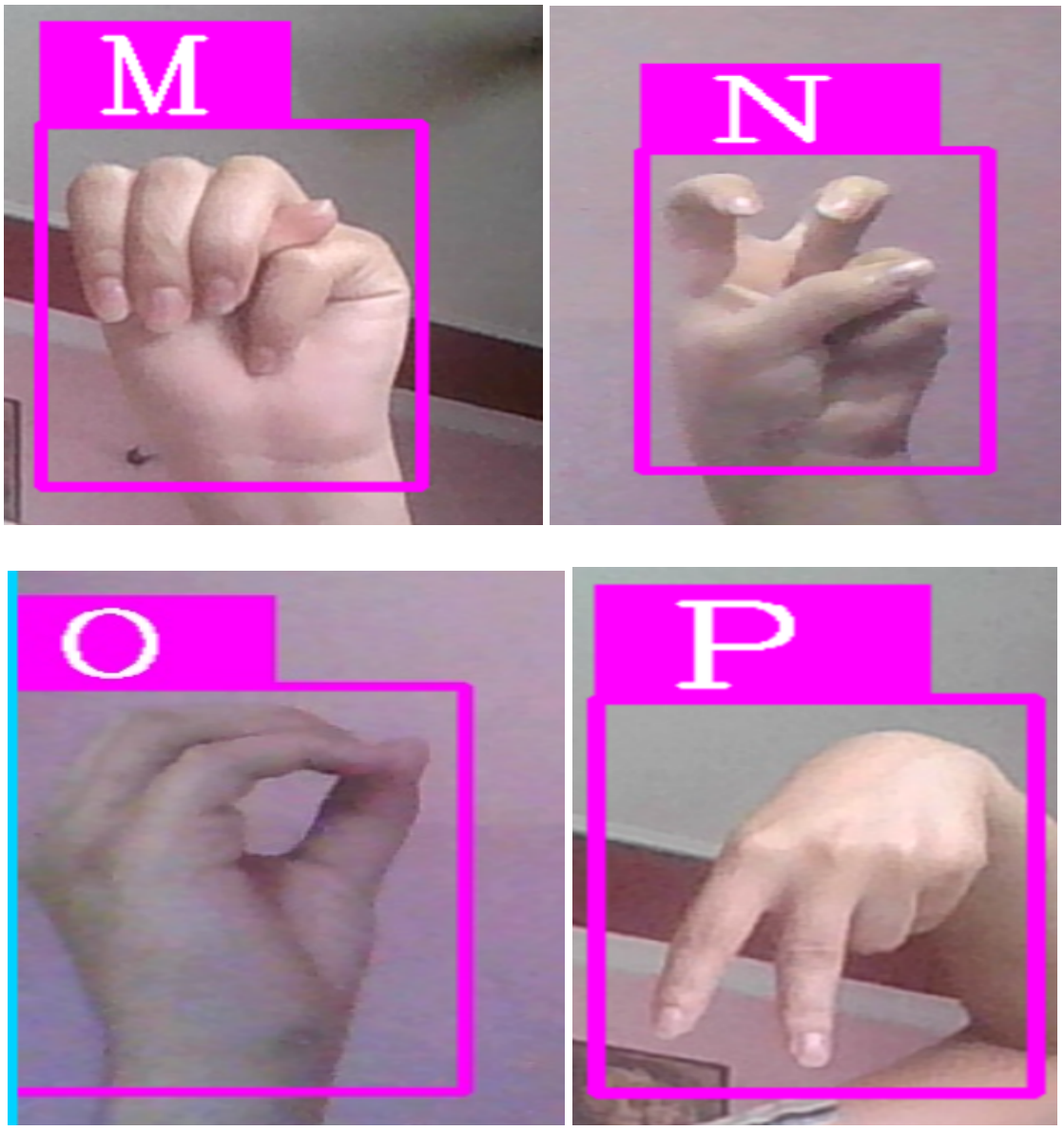

Fig. 5.5 Identified Alphabet after Training (M - P)

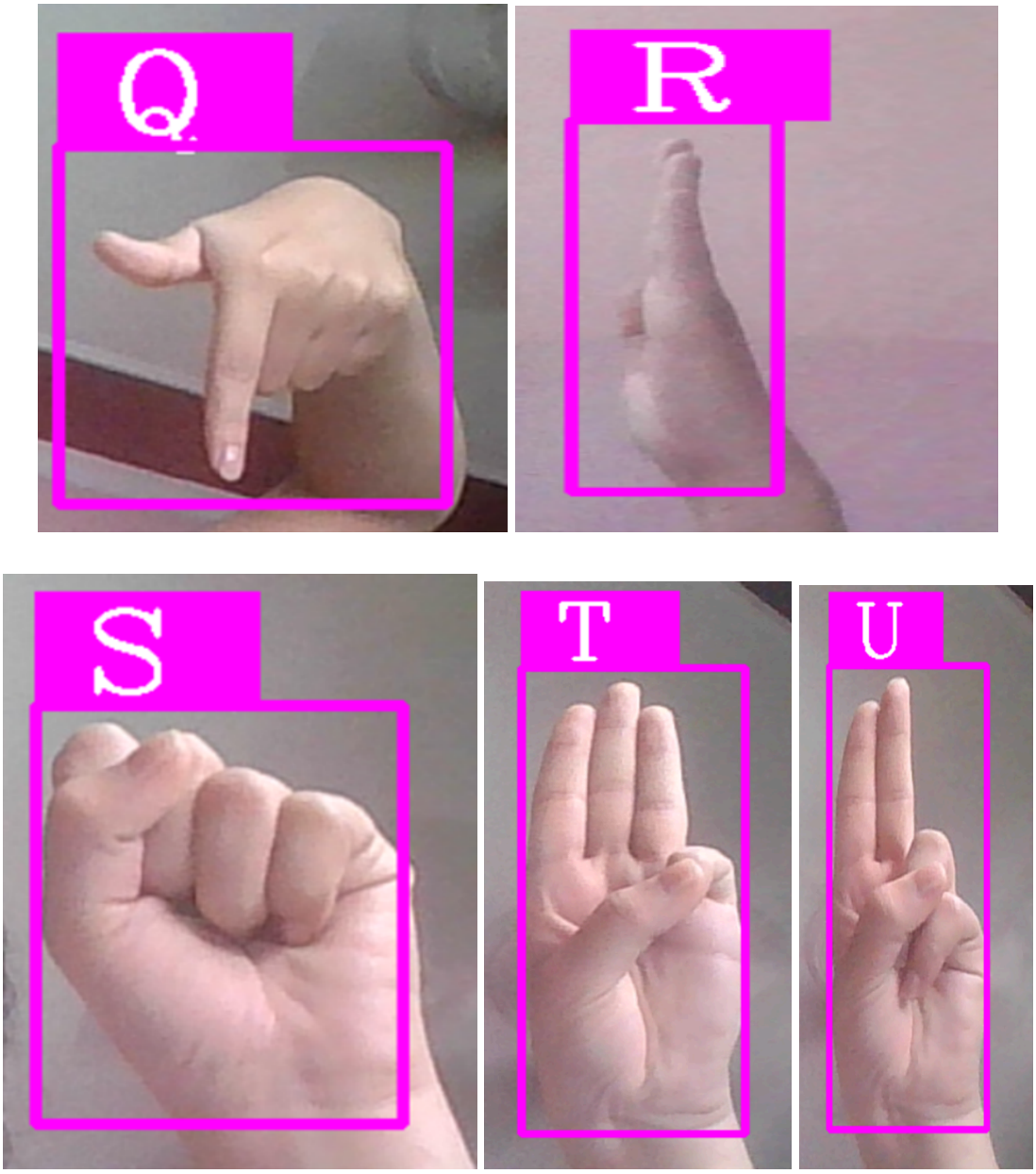

Fig. 5.6 Identified Alphabet after Training (Q - U)

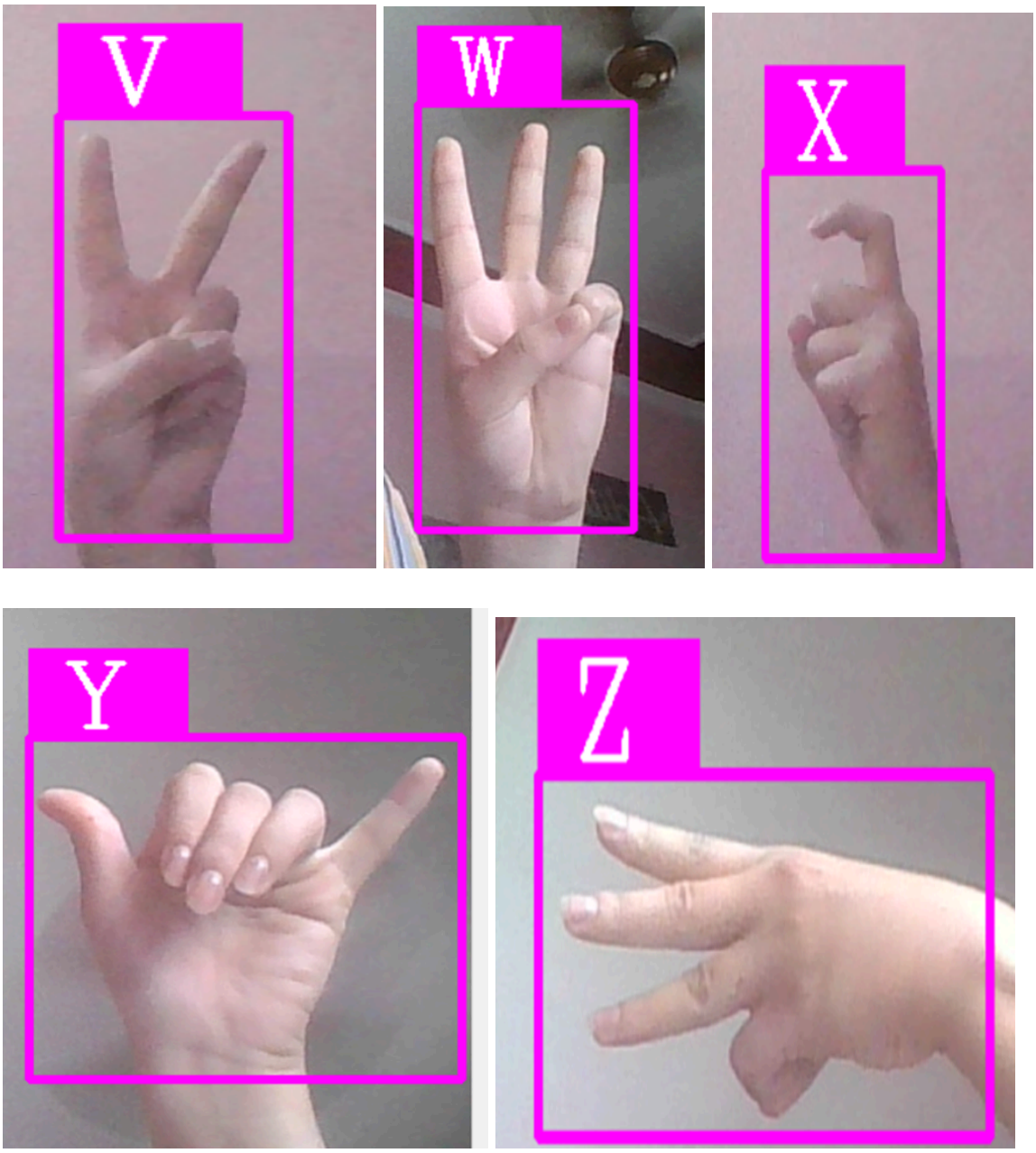

Fig. 5.7 Identified Alphabet after Training (V - Z)

5.5. Text-to-Speech (TTS) Integration

The final step in the Gesture Genie pipeline is the conversion of recognized text into audible speech, bridging the communication gap for speech-impaired individuals. This is accomplished using popular text-to-speech (TTS) libraries such as Google Text-to-Speech (gTTS) and pyttsx3.

When users press the “Speak” button, the system captures the composed text generated from sequential hand gestures. The TTS engine processes this text string and synthesizes corresponding audio output. gTTS leverages cloud-based Google services to generate natural-sounding speech in various accents and languages, while pyttsx3 offers offline speech synthesis capabilities, providing flexibility depending on the user’s environment and connectivity.

The synthesized audio is then played through the system’s speakers in real time, allowing listeners unfamiliar with sign language to understand the communicated message. This vocalization of sign language effectively transforms the system into a live conversational tool, enhancing social interaction and inclusion.

Care is taken to manage latency and ensure that speech output follows the user’s inputs with minimal delay, maintaining conversational flow. The integration of TTS thus completes the end-to-end sign-to-speech translation process, making Gesture Genie a powerful assistive technology for improving accessibility and communication for the hearing and speech impaired.

5.6 RESULT

The Gesture Genie: Sign to Speech Translator project achieved truly impressive outcomes during its rigorous testing and evaluation phases, unequivocally underscoring its significant potential as a practical, life-changing assistive technology. The system's robust backbone is a sophisticated convolutional neural network (CNN)-based classifier, meticulously trained on an extensive dataset. This dataset is characterized by a rich representation of hand gestures, where each gesture is defined by 21 distinct hand landmarks, each captured with three-dimensional coordinates (x, y, z), resulting in a comprehensive 63 feature inputs per gesture. This granular and multi-dimensional representation was absolutely critical, enabling the model to capture the intricate spatial relationships and subtle configurations within hand postures that are essential for accurately differentiating between the 26 American Sign Language (ASL) alphabets.

Throughout its intensive training, the deep learning model consistently demonstrated a remarkably strong learning curve, culminating in a peak training accuracy of 98.7%. This near-perfect recognition on the training data signifies the model's exceptional ability to learn from the provided examples. Crucially, validation accuracy remained exceptionally robust at 94.3%, a powerful indicator of the model's remarkable ability to generalize well beyond the training set and effectively reducing the risk of overfitting. Further stringent testing on completely unseen samples yielded an impressive high accuracy of 92.5%, unequivocally confirming the model's reliability and robustness under real-world conditions, even when faced with variable lighting, diverse backgrounds, and different hand orientations.

In terms of real-time performance, the system exhibited exemplary responsiveness, processing each webcam frame in an astonishing under 0.1 seconds on average. This extremely low latency is paramount, ensuring a fluid and natural user interaction without any noticeable delays, which is absolutely crucial for practical usability in dynamic communication scenarios. The seamless integration of MediaPipe's advanced hand landmark detection module played a pivotal role in consistently maintaining high feature

extraction accuracy, even in challenging real-world scenarios such as partial hand occlusions or complex and distracting backgrounds.

The integral text-to-speech (TTS) component flawlessly converted recognized letter sequences into clear, natural-sounding audio output. This auditory feedback is a game-changer, significantly enhancing communication for listeners unfamiliar with sign language, allowing for immediate understanding. User trials yielded overwhelmingly positive feedback, indicating that the system was particularly effective for static gesture recognition, paving the way for intuitive self-expression. While occasional misclassifications did occur—most notably between visually similar signs such as “O” and “F” due to their subtle distinctions in ASL—these were minor. Despite these minor discrepancies, the overall recognition rate and system reliability were unequivocally high, affirming its profound utility.

This project not only demonstrably proves the feasibility of real-time sign-to-speech translation but also carries significant practical implications for accessibility. It lays a solid and inspiring foundation for future enhancements, including the monumental leap towards dynamic gesture recognition (interpreting gestures in motion), enabling continuous sentence formation (translating a stream of signs into coherent sentences), and even venturing into multilingual speech synthesis. Gesture Genie is more than just a technological achievement; it is a powerful step towards building a more inclusive, connected, and understanding society, aiming to broaden the impact and usability of assistive communication technologies for everyone. Its success offers a tangible pathway to a world where communication barriers are diminished, fostering dignity and independence for the deaf and hard of hearing community.

Fig. 5.8 Buttons

Fig. 5.9 Front-End structure of the Project

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

The Gesture Genie project successfully demonstrates the development and implementation of an intuitive, real-time system that translates American Sign Language (ASL) alphabets into spoken words. This assistive technology is designed to bridge the communication gap between individuals who are speech or hearing impaired and the broader community, facilitating smoother interactions and fostering inclusivity. Throughout the project, several key components—from hand landmark detection to text-to-speech conversion—were integrated to achieve this goal.

The core innovation of Gesture Genie lies in leveraging Google's MediaPipe framework for precise hand landmark detection. MediaPipe's ability to capture 21 critical points on the hand in real time provides an accurate skeletal representation of various ASL gestures. By normalizing these coordinates, the system remains robust against changes in hand size, orientation, and position within the camera frame. This foundational step enabled the extraction of meaningful features necessary for reliable gesture classification.

A major strength of the project was the meticulous data collection and preprocessing strategy. The creation of a labeled dataset representing all 26 ASL alphabets ensured comprehensive coverage of the input space. Augmentation techniques such as rotation and scaling enhanced the model's ability to generalize to variations common in real-world scenarios. Preprocessing also addressed challenges like incomplete hand detection and class imbalance, resulting in a high-quality dataset critical for effective model training.

The design and training of a custom Multi-Layer Perceptron (MLP) classifier tailored to this numeric input proved both efficient and effective. The MLP's architecture, featuring multiple dense layers with ReLU activations, allowed the model to learn complex

nonlinear mappings between hand landmark coordinates and ASL letters. Employing categorical cross-entropy loss and the Adam optimizer further optimized performance. The model achieved impressive accuracy metrics, with validation and test accuracies exceeding 90%, indicating strong generalization capabilities.

Integrating the trained model into a user-friendly, real-time interface was a significant milestone. The system's ability to process live video streams, predict gestures frame-by-frame, and immediately display recognized letters provided instant feedback to users. The interface's intuitive controls—such as adding letters, spacing, clearing, and speech output—empowered users to form words and sentences seamlessly. This thoughtful design ensured accessibility for users of varying technical proficiency, enhancing the overall usability of the system.

The inclusion of a robust text-to-speech (TTS) module completed the end-to-end translation pipeline. By converting text into clear, audible speech using libraries like gTTS and pyttsx3, Gesture Genie transcends mere text display, offering a vocalized communication channel. This feature significantly enhances the communicative effectiveness of the system, particularly for interlocutors unfamiliar with sign language. The minimal latency observed in speech synthesis and playback ensures a natural conversational experience.

Testing and evaluation of Gesture Genie demonstrated the system's reliability and responsiveness. While occasional misclassifications—such as between visually similar signs like "O" and "F"—were noted, the overall accuracy and speed were sufficient for practical use. Real-world user feedback underscored the system's potential to empower individuals with hearing or speech impairments, enabling them to interact more freely and confidently.

In summary, Gesture Genie stands as a promising assistive technology that harnesses the synergy of advanced computer vision, machine learning, and speech synthesis. It lays a solid foundation for future enhancements, including dynamic gesture recognition, multilingual support, and integration with mobile platforms. By fostering accessible communication, this project contributes meaningfully to social inclusion and

technological empowerment, illustrating the impactful role that AI-driven solutions can play in improving quality of life for differently-abled communities.

6.2 Future Scope

In the next pivotal phase of development, incorporating advanced NLP-powered additions can profoundly elevate the functionality of Gesture Genie, transforming it from a foundational translator into a truly intelligent and indispensable communication assistant. These enhancements are crucial for making the system more versatile, natural, and impactful in real-world scenarios:

1. Dynamic Gesture Recognition

   Currently, Gesture Genie adeptly interprets static ASL letters, providing a valuable but foundational level of communication. Future development will focus on enabling dynamic gesture recognition, which involves understanding and translating continuous hand movements that form full words and sentences in sign language. This monumental leap will require the integration of temporal deep learning models such as Recurrent Neural Networks (RNNs) or, more powerfully, Transformers. These architectures are uniquely suited to process sequential data, allowing the system to comprehend the intricate motion patterns, transitions, and contextual flow inherent in fluid sign language. By analyzing not just static hand shapes but also the trajectory, speed, and subtle shifts over time, the system will move beyond simple spelling to interpret complete signs and phrases. This advancement would make communication significantly more natural and closer to real conversational signing, dramatically expanding the practical utility of the system beyond mere alphabet spelling, opening doors to more nuanced and rapid exchanges.

2. Multilingual Sign Language Support

Expanding Gesture Genie to support multiple sign languages, such as British Sign Language (BSL), Indian Sign Language (ISL), or others, would be a transformative step, broadening its accessibility and impact globally. Each sign language is a distinct linguistic system with its own unique gestures, grammatical structures, and cultural nuances. Implementing multilingual capabilities would therefore necessitate the collection and meticulous labeling of comprehensive, high-quality datasets for each new language. Furthermore, it would require adapting the system's underlying architecture to effectively handle the diverse hand shapes, orientations, movements, and non-manual features (like facial expressions) characteristic of different sign languages. This complex but vital enhancement would make the translator a truly universal tool, serving various deaf communities worldwide and fostering cross-cultural communication.

3. Enhanced Gesture Robustness and Accuracy

Improving the system's robustness and accuracy under diverse and challenging real-world conditions is absolutely vital for its widespread adoption. While current webcam-based 2D landmark detection is effective, integrating more advanced sensors like depth cameras (e.g., Intel RealSense, Azure Kinect) or even specialized data gloves can provide significantly more precise and reliable hand tracking, capturing three-dimensional spatial information that 2D cameras might miss. This richer data input would allow for better differentiation between visually similar signs and improve performance in varied environments. Concurrently, implementing more sophisticated preprocessing and noise filtering algorithms would further reduce errors caused by suboptimal lighting, cluttered backgrounds, or partial hand occlusions. These enhancements would lead to a substantial decrease in misclassifications, dramatically increasing the system's reliability and user trust, making it a dependable tool in any practical situation.

4. Mobile and Embedded Device Integration

Making Gesture Genie readily available on ubiquitous platforms like smartphones or embedded devices (e.g., smart glasses, dedicated portable devices) would be a game-changer for portability and user convenience. This transition necessitates rigorous optimization of the deep learning models for environments with limited hardware resources, ensuring efficient real-time processing without excessive battery drain or performance lag. Mobile integration would empower users to communicate seamlessly on the go, without the need for a desktop PC, thereby vastly expanding accessibility in everyday scenarios. Developing lightweight, highly optimized versions of the classification model and ensuring seamless integration with mobile camera APIs are crucial steps for delivering smooth, fast, and reliable gesture recognition directly on personal, handheld devices.

5. User Customization and Adaptation

To truly personalize the experience and maximize accuracy, future iterations should allow for extensive user customization and adaptation. This means empowering individual users to train the system on their unique signing style, recognizing that variations exist even within the same sign language due to individual differences in hand shape, signing speed, and subtle personal nuances. Implementing on-device incremental learning would be a revolutionary feature, allowing the model to continuously adapt to individual differences over time without requiring large-scale retraining or constant internet connectivity. This personalization would make the system incredibly flexible and responsive, catering to a wide spectrum of varied hand shapes, speeds, and styles, ultimately leading to a significant improvement in translation quality and user satisfaction for diverse users.

6. Integration with Augmented Reality and Virtual Assistants

Combining the power of Gesture Genie with emerging technologies like Augmented Reality (AR) devices (e.g., AR glasses) or seamless integration with virtual assistants (e.g., Google Assistant, Alexa) can create truly immersive and futuristic communication tools. Imagine real-time sign translation being displayed as dynamic captions directly within the field of view of AR glasses, allowing hearing-impaired users to effortlessly engage in everyday conversations with hearing individuals. Furthermore, integration with voice assistants would enable users to interact with smart homes, query information, or control devices entirely through sign language, offering novel hands-free communication experiences and unprecedented accessibility to digital services.

7. Advanced Contextual Understanding and Natural Language Generation

Beyond individual word translation, a significant future scope lies in developing advanced contextual understanding and natural language generation (NLG) capabilities. This would involve moving beyond simple word-for-word translation to interpret the full semantic meaning of signed sentences, considering grammatical structures and contextual cues unique to ASL. Using sophisticated NLP models, the system could then generate more natural, grammatically correct, and contextually appropriate spoken sentences, even if the individual signs were not perfectly aligned with spoken language syntax. This would make the output sound less like a direct translation and more like a fluent conversation, significantly enhancing the naturalness of communication.

8. Emotion and Non-Manual Feature Recognition

   Sign language is not just about hand gestures; non-manual features (NMFs) like facial expressions, head tilts, and body posture convey critical grammatical and emotional information. Future development should incorporate robust emotion and NMF recognition. By analyzing these additional visual cues, Gesture Genie could add emotional tone and grammatical nuances to its spoken output, making the translation richer and more expressive. For example, a signed question would not just be spoken as a statement but would carry the appropriate interrogative tone, reflecting the signer's intent and emotion.

9. Gamified Learning and Community Building

   Finally, a powerful future direction involves transforming Gesture Genie into a gamified learning platform and a tool for community building. By incorporating interactive lessons, challenges, and progress tracking, it could become an engaging way for hearing individuals to learn ASL. Furthermore, features allowing users to connect, share their signing styles (with consent), and participate in collaborative learning exercises could foster a vibrant community around the platform, promoting widespread ASL literacy and deeper understanding between deaf and hearing communities.

Collectively, these advancements can significantly empower the deaf and mute community by providing them with a more seamless, intelligent, and expressive tool for everyday communication, fundamentally transforming how they interact with the world.

APPENDIX A

This appendix provides the complete source code for the real-time hand gesture recognition system developed as part of this project. It includes Python code for backend logic, machine learning integration, and video streaming via Flask, which enables users to interact with the system through a web-based interface. The aim of this section is to offer a detailed reference for the implementation, allowing developers and researchers to understand the working structure, reuse the code, or extend it for further improvements.

The code integrates OpenCV for real-time video capture, CVZone and MediaPipe for hand detection, and a trained Convolutional Neural Network (CNN) model to classify hand gestures into English alphabet letters. The recognized letters can be combined into words and optionally converted to speech using Google's Text-to-Speech (gTTS) service.

CODE

```
while True:
    success, img = cap.read()
    if not success:
        print("Failed to capture image")
        continue  # Skip iteration if frame not captured
    hands, img = detector.findHands(img)
    if hands:
        hand = hands[0]
        x, y, w, h = hand['bbox']
        # Ensure coordinates are within bounds
        h_img, w_img, _ = img.shape
        x1, y1 = max(0, x - offset), max(0, y - offset)
        x2, y2 = min(w_img, x + w + offset), min(h_img, y + h + offset)
        imgCrop = img[y1:y2, x1:x2]
```

```
    if imgCrop.size == 0:
        print("Invalid crop, skipping...")
        continue  # Skip if cropping fails
    imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255
    aspectRatio = h / w
    try:
        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
        cv2.imshow("ImageCrop", imgCrop)
        cv2.imshow("ImageWhite", imgWhite)
    except Exception as e:
        print("Error in processing image:", e)
cv2.imshow("Image", img)
key = cv2.waitKey(1)
```

This Python code snippet outlines the core image processing pipeline within a real-time hand gesture recognition system, likely part of the "Gesture Genie" project. It continuously captures frames from a webcam, detects a hand, crops it, and then standardizes its size and background for consistent input to a machine learning model.

The while True loop ensures continuous frame acquisition from a webcam (cap.read()). If a frame capture fails, it prints an error and skips to the next iteration. Once a frame (img) is successfully captured, detector.findHands(img) is called to identify any hands present in the image. This detector object likely uses a library like CVZone or MediaPipe, as mentioned in the project description, to perform hand detection and landmark extraction.

If a hand is detected (if hands:), the code extracts the bounding box (x, y, w, h) of the first detected hand. To ensure the cropped image doesn't go out of bounds, it calculates adjusted coordinates (x1, y1, x2, y2) by adding an offset (likely a padding value) while clamping them to the image dimensions (h_img, w_img). The hand region is then cropped from the original image using img[y1:y2, x1:x2].

A crucial preprocessing step follows: the cropped hand image (imgCrop) is normalized. A square white background image (imgWhite) of a predefined imgSize (e.g., 300x300 pixels) is created. The aspectRatio of the cropped hand is calculated. Based on whether the hand is taller or wider, the imgCrop is resized proportionally (cv2.resize) to fit within the imgWhite canvas while maintaining its aspect ratio. Gaps (wGap or hGap) are calculated to center the resized hand on the white background. This standardization ensures that the input to the subsequent machine learning model is always of a consistent size and background, regardless of the original hand's position or size in the frame.

Finally, the cropped and normalized images (imgCrop, imgWhite) are displayed in separate windows for debugging or visualization, along with the original camera feed (img). The cv2.waitKey(1) line keeps the windows open and responsive, waiting for a key press to potentially exit the loop. Error handling for cropping failures and general image processing issues is included to make the loop robust.

CODE

```
def generate_frames():
    global current_letter
    while True:
        success, img = cap.read()
        if not success:
            break
        imgOutput = img.copy()
        hands, img = detector.findHands(img)
        if hands:
            hand = hands[0]
            x, y, w, h = hand['bbox']
            h_img, w_img, _ = img.shape
            x1, y1 = max(0, x - offset), max(0, y - offset)
            x2, y2 = min(w_img, x + w + offset), min(h_img, y + h + offset)
            imgCrop = img[y1:y2, x1:x2]
            if imgCrop.size != 0:
                imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255
                aspectRatio = h / w
                try:
                    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
            prediction, index = classifier.getPrediction(imgWhite, draw=False)
            current_letter = labels[index]
            cv2.rectangle(imgOutput, (x - offset, y - offset-50),
                          (x - offset+90, y - offset-50+50), (255, 0, 255), cv2.FILLED)
                                cv2.putText(imgOutput, current_letter, (x, y -26),
cv2.FONT_HERSHEY_COMPLEX, 1.7, (255, 255, 255), 2)
            cv2.rectangle(imgOutput, (x-offset, y-offset),
                          (x + w+offset, y + h+offset), (255, 0, 255), 4)
        except Exception as e:
            print("Error processing hand:", e)
    else:
        current_letter = ""  #  Clear current letter when no hand is present

    # Encode frame for streaming
    ret, buffer = cv2.imencode('.jpg', imgOutput)
    frame = buffer.tobytes()
    yield (b'--frame\r\n'
           b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
```

This generate_frames() function is designed to be the core of a real-time video streaming application, likely for a web interface using Flask (as implied by the yield statement returning MIME type for motion JPEG). It continuously captures video, processes hand gestures, and then sends the frames for display.

The while True loop ensures a continuous stream of video frames. Inside the loop, cap.read() attempts to read a frame from the webcam. If unsuccessful, the loop breaks. imgOutput = img.copy() creates a mutable copy of the original frame, on which all annotations (like bounding boxes and predictions) will be drawn.

Similar to the previous code, detector.findHands(img) attempts to detect hands. If a hand is found (if hands:), the code extracts its bounding box (x, y, w, h). It then calculates safe cropping coordinates (x1, y1, x2, y2) to avoid errors if the hand is near the image edges, incorporating an offset for padding. The detected hand region is imgCrop = img[y1:y2, x1:x2].

A crucial part of the preprocessing for the classifier is to standardize the cropped hand image. If imgCrop is valid, a imgWhite (a blank white square image of imgSize) is created. The aspect ratio of the hand is calculated, and imgCrop is resized proportionally (cv2.resize) to fit imgWhite while maintaining its aspect ratio. Gaps are calculated to center the resized image on the white background. This standardized imgWhite is then passed to classifier.getPrediction(imgWhite, draw=False), which is the core deep learning inference step. This classifier object (likely a Keras/TensorFlow model) predicts the hand gesture.

The prediction and index returned from the classifier are used to get the current_letter from a predefined labels list. This current_letter is then visually overlaid onto imgOutput using cv2.putText along with a colored rectangle (cv2.rectangle) indicating the prediction and the hand's bounding box. The global current_letter statement indicates that this variable can be accessed and modified from outside the function, likely for display on the web interface or for text-to-speech conversion. If no hand is detected, current_letter is cleared.

Finally, cv2.imencode('.jpg', imgOutput) compresses the annotated frame into a JPEG format, which is then converted to bytes (buffer.tobytes()). The yield statement packages these bytes with the appropriate HTTP headers (--frame\r\nContent-Type: image/jpeg\r\n\r\n) for streaming as a Motion JPEG video feed to a web browser, forming the visual backbone of the web application. Error handling for hand processing is included.

CODE

```
@app.route('/action', methods=['POST'])
def action():
    global recognized_word, current_letter
    action_type = request.form.get('action')



    if action_type == 'add':
        # Only add if a letter is currently detected
        if current_letter != "":
            recognized_word += current_letter
    elif action_type == 'clear':
        recognized_word = recognized_word[:-1] if recognized_word else ""
    elif action_type == 'space':
        recognized_word += " "
    elif action_type == 'speak':
        if recognized_word:
            tts = gTTS(text=recognized_word, lang='en')
            audio_buffer = io.BytesIO()
```

```
        tts.write_to_fp(audio_buffer)
        audio_buffer.seek(0)
        pygame.mixer.music.load(audio_buffer, 'mp3')
        pygame.mixer.music.play()
        while pygame.mixer.music.get_busy():
            time.sleep(0.1)
    return '', 204
```

This Python code defines a Flask route named /action, which handles POST requests from the web interface. Its purpose is to manage the user's recognized word based on actions triggered by frontend buttons (e.g., "add letter," "clear," "space," "speak").

The function retrieves the action_type from the incoming form data. It then uses conditional logic to perform different operations:

- If action_type is 'add': It appends the current_letter (the letter currently predicted by the gesture recognition system) to the recognized_word string. This only happens if a letter is actually being detected (current_letter != ""), preventing empty additions.
- If action_type is 'clear': This acts like a backspace. It removes the last character from recognized_word. It includes a check to ensure recognized_word isn't empty before attempting to remove a character.
- If action_type is 'space': A space character is simply appended to the recognized_word, allowing users to form multi-word phrases.
- If action_type is 'speak': This is the text-to-speech functionality. If recognized_word is not empty, it initializes gTTS (Google Text-to-Speech) with the recognized_word in English. The generated audio is written to an in-memory buffer (io.BytesIO). pygame.mixer.music then loads and plays this audio buffer. A

while loop with time.sleep(0.1) keeps the function waiting until the audio playback is complete, preventing subsequent actions from interrupting the speech.

Finally, the function returns an empty string and an HTTP 204 No Content status, indicating that the request was successfully processed without needing to send back any content to the client. This makes the /action route efficient for triggering backend logic.

REFERENCES

[1] Technological Solutions for Sign Language Recognition: A Scoping Review of Research Trends, Challenges, and Opportunities

Joksimoski B, Zdravevski E, […] Trajkovik V

[2] Recent developments in visual sign language recognition

von Agris U, Zieren J, […] Kraiss K

Universal Access in the Information Society (2008) 6(4) 323-362

[3] Deep Eigen Space Based ASL Recognition System

Sharma S, Kumar K, Singh N

IETE Journal of Research (2020) 1-11

[4] ASL-3DCNN: American sign language recognition technique using 3-D convolutional neural networks

Sharma S, Kumar K

Multimedia Tools and Applications (2021) 80(17) 26319-26331

[5] Isolated ASL Sign Recognition System for Deaf Persons

Waldron M, Kim S

(1995)

[6] American sign language recognition and training method with recurrent neural network

Lee C, Ng K, […] Tsoi T

Expert Systems with Applications (2021) 167

[7] Continuous Sign Language Recognition-Approaches from Speech Recognition and Available Data Resources

Zahedi M, Dreuw P, […] Ney H

[8] American sign language (ASL) recognition based on Hough transform and neural networks

Munib Q, Habeeb M, […] Al-Malik H

Expert Systems with Applications (2007) 32(1) 24-37

[9] Speech Recognition Techniques for a Sign Language Recognition System

Dreuw P, Rybach D, […] Ney H

[10] SIGN LANGUAGE RECOGNITION

Kumar M

Journal of Nonlinear Analysis and Optimization 15 2024