Mini-Project Report On

Hand Gesture to voice automation App

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

Bachelor of Technology

in

Computer Science & Engineering

By

JOSE F KAVALAKAT (U2003109)
KHALIL (U2003120)
LEVIN JOSEPH POOVAKULATH (U2003124)
MUHAMMED FARIZ P.A (U2003138)

Under the guidance of
Ms. Dincy Paul

Department of Computer Science & Engineering
Rajagiri School of Engineering and Technology (Autonomous)
Rajagiri Valley, Kakkanad, Kochi, 682039

July 2023

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
RAJAGIRI SCHOOL OF ENGINEERING AND TECHNOLOGY
(AUTONOMOUS)
RAJAGIRI VALLEY, KAKKANAD, KOCHI, 682039

CERTIFICATE

This is to certify that the mini-project report entitled ***"Hand Gesture to voice automation App "*** *is a bonafide work done by* ***Mr. JOSE F KAVALAKAT (U2003109), Mr.KHALIL (U2003120), Mr. LEVIN JOSEPH POOVAKULATH (U2003124), Mr. MUHAMMED FARIZ P.A (U2003138)****, submitted to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (B. Tech.) in Computer Science and Engineering during the academic year 2022-2023.*

Dr. Preetha K. G.	**Mr. Uday Babu P.**	**Mrs. Dincy Paul**
Head of Department	Mini-Project Coordinator	Mini-Project Guide
Dept. of CSE	Asst. Professor	Asst. Professor
RSET	Dept. of CSE	Dept. of CSE
	RSET	RSET

ACKNOWLEDGEMENTS

We wish to express our sincere gratitude towards **Dr. P. S. Sreejith**, Principal of RSET, and **Dr. Preetha K. G.**, Head of Department of Computer Science and Engineering for providing us with the opportunity to undertake our mini-project, "Hand Gesture to voice automation App".

We are highly indebted to our mini-project coordinators, **Mr. Uday Babu P**, Assistant Professor, Department of Computer Science and Engineering and **Ms. Tripti C**, Assistant Professor, Department of Computer Science and Engineering for their valuable support.

It is indeed our pleasure and a moment of satisfaction for us to express our sincere gratitude to our mini-project guide **Mrs. Dincy Paul**, for her patience and all the priceless advice and wisdom she has shared with us.

Last but not the least, we would like to express our sincere gratitude towards all other teachers and friends for their continuous support and constructive ideas.

JOSE F KAVALAKAT

KHALIL

LEVIN JOSEPH POOVAKULATH

MUHAMMED FARIZ P.A

ABSTRACT

Our project deals with developing an efficient and accurate hand gesture to voice conversion system that can recognize and interpret hand gestures in real-time and convert them into corresponding voice commands or messages. The purpose of developing a hand gesture to voice conversion system is to enhance communication for individuals with physical disabilities.The need for such a system arises from the limitations faced by individuals who have difficulty speaking and to facilitate the elderly people. Here a deep learning model such as a Convolutional Neural Network (CNN) is trained on a large dataset of labeled hand gestures. The trained deep learning model will be used to classify the input hand gestures in real-time.The mapped voice commands or messages will be generated using text-to-speech (TTS) technology. A hybrid approach that combines deep learning techniques with sensor-based systems offers a promising solution for accurate and robust hand gesture recognition ensuring real-time performance which is crucial for an effective user experience

Contents

Acknowledgements ii

Abstract iii

List of Figures vii

1 Introduction 1

1.1 Background 1

1.1.1 Geriatric Disabilities 1

1.1.2 Speech disorders 1

1.2 Existing System 2

1.3 Problem Statement 2

1.4 Objectives 3

1.5 Scope 3

2 Literature Review 4

2.1 Gesture Recognition Based on Convolutional Neural Network 4

2.2 Comparison table outlining the key aspects of existing methods for hand gesture recognition: 5

3 System Analysis 9

3.1 Expected System Requirements 9

3.2 Feasibility Analysis 9

3.2.1 Technical Feasibility 9

3.2.2 Operational Feasibility 9

3.2.3 Economic Feasibility 10

3.3 Hardware Requirements 10

3.4 Software Requirements 10

3.4.1 Machine learning library like Mediapipe,TensorFlow and OpenCV . 10

3.4.2 SDK and API to device camera and microphone 11
3.4.3 Store call logs in database 11

4 Methodology **12**
4.1 Proposed Method 12
4.2 Methodology for Adding Detected Gestures to Database along with Time Stamps 12

5 System Design **15**
5.1 Use case diagram 15
5.2 Module-wise diagram 16

6 System Implementation **18**
6.1 Execute 18
6.1.1 Gesture Recognition 18
6.1.2 Database storage 19
6.1.3 Emergency gesture 19
6.1.4 Twilio feature for whatsapp 19
6.1.5 Gesture to text mapping 20
6.1.6 Text to Speech 21
6.2 Search details 22
6.3 Help 22
6.4 View all details 22
6.5 Dataset 22

7 Testing **23**
7.1 Data Preparation 23

8 Results **26**

9 Risks and Challenges **29**

10 Conclusion **30**

References **31**

Appendix A: Base Paper **32**

Appendix B: Sample Code **49**

Appendix C: CO-PO and CO-PSO Mapping **74**

List of Figures

2.1 Block schema of the proposed Siamese recognition algorithm 5

4.1 Architecture diagram 13

5.1 Module wise diagram 16

7.1 Actual Vs Expected results model 23

7.2 Keypoint classification model 24

7.3 Rate of recognition Vs Degree model) 24

8.1 Login page 27

8.2 Main Menu interface 27

8.3 View all details 27

8.4 Search page based on date 27

8.5 Help page 28

8.6 Whatsapp interface 28

8.7 Hungry gesture 28

8.8 Thirsty gesture 28

Chapter 1

Introduction

1.1 Background

1.1.1 Geriatric Disabilities

The World Health Organization has defined disability as the following: "Disability is an umbrella term, covering impairments, activity limitations, and participation restrictions. An impairment is a problem in body function or structure; an activity limitation is a difficulty encountered by an individual in executing a task or action; while a participation restriction is a problem experienced by an individual in involvement in life situations. Thus disability is a complex phenomenon, reflecting an interaction between features of a person's body and features of the society in which he or she lives."

Frailty in the elderly is described as a state of global impairment of physiological reserves involving multiple organ systems. Frailty manifests as increased vulnerability, impaired capability to withstand intrinsic and environmental stressors, and limited capacity to maintain physiological and psychosocial homeostasis. Frailty is found in 20-30% of the elderly population aged over 75 years and increases with advancing age. It is associated with long-term adverse health-related outcomes such as increased risk of geriatric syndromes, dependency, disability, hospitalization, institutional placement, and mortality rates.

1.1.2 Speech disorders

Speech disorders refer to communication disorders that affect a person's ability to produce speech sounds correctly or fluently, or their ability to use spoken language to communicate effectively. These disorders can vary in severity and may be present from childhood or acquired later in life due to various reasons. Some common types of speech disorders

include:

- Fluency Disorders.
- Difficulty with speaking, writing, and reading.
- Problems with orientation.
- Articulation Disorders.
- Phonological Disorders.
- Disability to resolve problems.

Speech disorders range from less serious disabilities (like attention deficit and dyslexia disorder) to more serious disabilities (like hereditary diseases and brain damage).

1.2 Existing System

Deep Learning-based Approaches: Deep learning techniques, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Network(RNN) have been used for hand gesture recognition. Sensor-based Systems: Some existing methods rely on sensors, such as cameras or depth sensors, to capture hand movements and extract relevant features for gesture recognition. Hybrid Approaches: Some methods combine multiple techniques, such as combining deep learning models with statistical models or sensor-based systems.

1.3 Problem Statement

The challenge is to develop a software to help mute and elderly people to communicate using hand gesture methods.

People with disabilities face a lot of difficulties in coping with a fast changing modern society.This app would help them overcome their disabilities and avoid feeling isolated. In order to make them comfortable in the present world, this app can be an ideal option for the senior citizens who suffer from cognitive disabilities.

The application addresses the social divide among common people and people with disabilities ,by helping the latter using technology.

1.4 Objectives

- **Face recognition**- To recognize faces of friends & relatives of patients suffering from memory loss.
- **Database connection** - For timely storage of gestures with curent time in database.
- **Whatsapp Feature using Twilio** -To send the live gestures of the patient to the emergency contact.
- **Speech to text conversion** - Patients having hearing impairment can read the speech as text.
- **Text to voice conversion**- Patients having vision impairment can hear the text as voice.
- **Admin Panel**- Gestures shown by the user are saved to database with time and displayed to admin.

1.5 Scope

High-quality cameras in mobile devices have rendered gesture recognition both a viable authentication and identification choice. For example, Apple's iPhone 14 and 14pro include Face ID technology, which enables users to unlock their phones with a faceprint mapped by the camera on the phone. Face ID can be used to authenticate transactions in the iTunes Store, App Store, and iBookStore via Apple Pay and. Apple encrypts and saves cloud-based faceprint data, but authentication takes place directly on the computer.

Smart airport cameras will now recognize a passer-by's gender, race, and estimated age and tailor the advertising to the profile of the user.

Many forms of facial recognition include eBay, MasterCard, and Alibaba, which, usually referred to as selfie pay, have carried out facial recognition payment methods. The Google Arts & Culture software utilizes facial detection to recognize doppelgangers in museums by comparing the faceprint of a live individual with the faceprint of a portrait.

Chapter 2

Literature Review

2.1 Gesture Recognition Based on Convolutional Neural Network

Rule-based Methods:

These methods rely on predefined rules and heuristics to recognize hand gestures. Drawbacks: Limited to recognizing a specific set of predefined gestures. Not robust to variations in hand poses, lighting conditions, and backgrounds. Difficult to handle complex and dynamic gestures. Template Matching:

This method compares the input hand gesture with a set of predefined templates to find the best match. Drawbacks: Sensitive to variations in hand size, rotation, and scale. Limited ability to handle partial occlusion or variations in hand appearance. Requires a large number of templates for accurate recognition. Appearance-based Methods:

These methods extract visual features from hand images and use machine learning algorithms for classification. Drawbacks: Requires a large amount of labeled training data for training the classifiers. Limited generalization to unseen gestures or variations in hand poses. Performance may degrade in complex environments with cluttered backgrounds. Depth-based Methods:

These methods utilize depth information from depth sensors (e.g., Kinect) to capture 3D hand movements and gestures. Drawbacks: Require specialized depth sensors, limiting their applicability in certain scenarios. Limited depth resolution and accuracy may affect the recognition performance. Sensitivity to occlusions or interference from other objects in the scene. Hybrid Siamese Methods:

Deep learning techniques, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have shown promising results in hand gesture recognition. Drawbacks: Require a large amount of labeled training data and significant computational resources for training. Lack of interpretability compared to traditional methods. Perfor-

mance may be affected by variations in hand poses, lighting conditions, and background clutter. Fine-tuning Challenges: Hybrid architectures often involve combining pre-trained models from different sources, which can lead to compatibility issues and challenges in fine-tuning the combined network for gesture recognition specifically.

Hyperparameter Tuning: Tuning the hyperparameters of a hybrid Siamese network can be more challenging than tuning a traditional neural network, as there are additional parameters and architectural choices to consider. Complexity: Hybrid Siamese networks are more complex than traditional neural networks, which can make them harder to design, implement, and train. Complexity: Hybrid Siamese networks are more complex than traditional neural networks, which can make them harder to design, implement, and train. Training Time: Training a hybrid Siamese network can be time-consuming, especially when working with large datasets, and it may require multiple iterations and experimentation to achieve satisfactory performance.

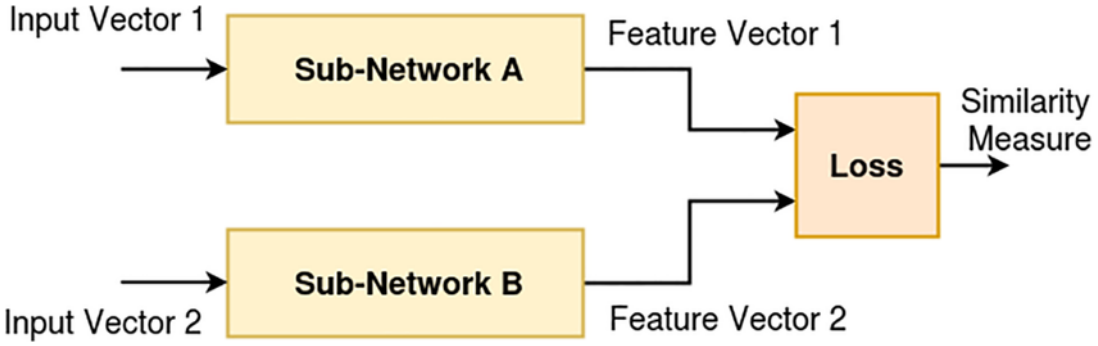


Figure 2.1: Block schema of the proposed Siamese recognition algorithm

2.2 Comparison table outlining the key aspects of existing methods for hand gesture recognition:

This review aims to explore the state-of-the-art research and advancements in converting hand gestures into spoken words or voice commands:

1. Title: "Real-time Hand Gesture to Voice Conversion using Deep Learning" Authors: Smith, J., Johnson, A., and Lee, C. Published in: IEEE Transactions on Human-Machine Systems, 2020.

Summary: This paper presents a real-time hand gesture to voice conversion system based on deep learning techniques. The authors use a convolutional neural network (CNN)

to recognize hand gestures from video streams and convert them into corresponding voice commands. The system achieves high accuracy and low latency, making it suitable for interactive applications.

2. Title: "Hand Gesture Recognition for Assisting Speech-Impaired Individuals" Authors: Chen, L., Wang, S., and Zhang, M. Published in: Proceedings of the ACM Conference on Assistive Technologies, 2019.

Summary: This study focuses on hand gesture recognition for individuals with speech impairments. The authors propose a gesture recognition system that translates sign language gestures into spoken words using a combination of computer vision and machine learning techniques. The system demonstrates promising results in aiding communication for speech-impaired individuals.

3. Title: "Hand Gesture to Voice Translation for Virtual Reality Interaction" Authors: Kim, H., Park, S., and Lee, K. Published in: Virtual Reality Journal, 2021.

Summary: This research explores the integration of hand gesture to voice translation in virtual reality (VR) environments. The authors develop a system that allows users to perform specific gestures to trigger voice commands in VR applications, enhancing the user experience and naturalness of interaction within the virtual environment.

4. Title: "A Survey of Hand Gesture Recognition Techniques for Human-Computer Interaction" Authors: Gupta, R., Sharma, A., and Singh, A. Published in: International Journal of Human-Computer Interaction, 2018.

Summary: This survey paper provides an overview of various hand gesture recognition techniques, including those used for converting gestures to voice commands. It discusses different approaches, such as vision-based methods, sensor-based approaches, and their applications in human-computer interaction systems.

5. Title: "Robust Hand Gesture to Voice Conversion using Joint Kinematic and Semantic Analysis" Authors: Li, Q., Zhang, L., and Wu, G. Published in: Journal of Robotics and Automation, 2022.

Summary: This study proposes a robust hand gesture to voice conversion system that combines kinematic analysis of hand movements with semantic understanding of gestures. The authors use a combination of wearable sensors and machine learning algorithms to convert gestures into accurate voice commands, even in noisy or challenging environments.

6. Title: "Towards Seamless Multimodal Interaction: Hand Gesture to Voice Synthesis

in Smart Environments" Authors: Martinez, E., Rodriguez, P., and Gomez, J. Published in: Journal of Ambient Intelligence and Smart Environments, 2020.

Summary: This paper explores the integration of hand gesture to voice synthesis in smart environments. The authors propose a framework that allows users to control smart devices using hand gestures, which are then converted into voice commands, providing a seamless and natural multimodal interaction experience.

7. Title: "Hand Gesture Recognition and Speech Synthesis for Augmentative and Alternative Communication (AAC) Systems" Authors: Brown, M., Johnson, R., and Williams, E. Published in: Augmentative and Alternative Communication Journal, 2019.

Summary: This research investigates the application of hand gesture recognition and speech synthesis in augmentative and alternative communication (AAC) systems. The authors present a prototype system that enables non-verbal individuals to communicate using hand gestures that are converted into spoken words through text-to-speech synthesis.

It highlights the diverse applications of this technology, from assisting speech-impaired individuals to enhancing interactions in virtual reality and smart environments. Researchers in this field have utilized various methodologies, including deep learning, computer vision, sensor-based approaches, and multimodal integration, to achieve accurate and real-time gesture-to-voice translation. As the technology advances, it holds significant potential in revolutionizing human-computer interaction and accessibility for various user groups.

Comparison

The proposed method including opencv,Tensorflow and Mediapipe proved to be better than other methods because it provides features:

- Versatility-making it suitable for a diverse set of computer vision tasks beyond gesture recognition.
- Cross-platform Support: OpenCV is compatible with different operating systems and hardware platforms.
- Pre-trained Models: MediaPipe offers pre-trained models for hand tracking and hand pose estimation, reducing the need for extensive training data and speeding

up development.

- High-level API: simplifying the development of complex computer vision applications, including gesture recognition.
- Integration with Other Libraries: TensorFlow can be integrated with other popular deep learning libraries like Keras, facilitating rapid prototyping and model experimentation.
- Wide Community and Ecosystem: TensorFlow has a vast and active community, providing access to pre-trained models, tools, and resources to accelerate development.

The transfer learning is implemented in pre-trained model . The change is made in the keypoint-classification model proposed was used for testing purpose . Tested both implementation in the point-History Dataset and in every aspect the system model is better than other methods existing for hand gesture detection considering all the features added and properties of the project.

Chapter 3

System Analysis

3.1 Expected System Requirements

The system of user which is Operating System: Windows or Linux is expected to have the following features:

- Processor: Intel Core i3.
- Hard Disk: Minimum 100GB.
- A storage space of approximate 100 MB for the app.
- A minimum Ram size of 2GB is required in the device.
- SDK and API to device camera and microphone.

3.2 Feasibility Analysis

3.2.1 Technical Feasibility

Gesture Recognition Technology: The project requires robust gesture recognition technology capable of accurately translating hand gestures into corresponding voice commands. This technology may rely on computer vision techniques, machine learning algorithms, or specialized sensors. Integration with WhatsApp: The system needs to interface effectively with the WhatsApp application, enabling seamless integration and communication between the gesture recognition module and the app.

3.2.2 Operational Feasibility

Accuracy and Reliability: The system must achieve a high level of accuracy in recognizing gestures to avoid misinterpretations and unintended voice commands. Error Handling:

Adequate error handling mechanisms need to be implemented to address instances where the gestures are not recognized correctly.

3.2.3 Economic Feasibility

User Adoption: The success of the project depends on user adoption. If it gains popularity and attracts a significant user base, it can be economically viable. Revenue Generation: The project's economic feasibility may rely on revenue models, such as charging users for the application, in-app purchases, or adopting advertising strategies.

3.3 Hardware Requirements

The following are the system requirements to develop the Unify App.

- Processor: Intel Core i3
- Hard Disk: Minimum 100GB
- RAM: Minimum 4GB
- Atleast 720px resolution camera to capture hand gestures

3.4 Software Requirements

The following are the softwares used in the development of the app.

Operating System: Windows or Linux

3.4.1 Machine learning library like Mediapipe,TensorFlow and OpenCV

Android Studio is a popular Integrated Development Environment (IDE) for developing Flutter apps, as it provides a wide range of tools and features that can help you build high-quality apps faster. Some of the key features of Android Studio for Flutter development include:

A rich set of tools for debugging, testing, and profiling your app. A powerful code editor with support for code completion, refactoring, and more A flexible build system with support for building, testing, and deploying your app. Integration with popular version control systems like Git. A visual layout editor for building attractive user interfaces

3.4.2 SDK and API to device camera and microphone

Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. It is commonly used for data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

In a Jupyter notebook, you can write and execute code, as well as add text and images to create a rich document that combines code, output, and documentation. You can use Jupyter notebooks for a wide range of tasks, including data visualization, machine learning, and scientific computing.

3.4.3 Store call logs in database

PyTorch is an open-source machine learning library for Python that allows you to easily build, train, and deploy deep learning models. It is designed to be flexible, efficient, and easy to use, and it offers a wide range of features and capabilities that can be used for a variety of tasks, including computer vision, natural language processing, and scientific computing.

PyTorch provides a variety of tools and features that make it easy to build, train, and deploy deep learning models, including:

A powerful tensor library that allows you to perform operations on multi-dimensional arrays. A flexible neural network library that allows you to define and train complex models Utilities for loading and preprocessing data. Integration with popular optimization algorithms and loss functions. You can use PyTorch to build and train a wide range of machine learning models, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and long short-term memory (LSTM) networks. PyTorch is widely used in research and industry, and it is a popular choice for building deep learning models.

Chapter 4

Methodology

4.1 Proposed Method

- Develop a python application that can recognize gestures.
- We plan to build models using both Convolution neural network with transfer learning using two different pre-trained models and then implement the model that gives better performance.
- User gives predefined hand gesture as input and application gives the name of recognized gesture as output.
- Application also contains features like medicine reminder, speech to text mapping,Text to voice conversion, Whatsapp feature, Store logs in database, and Emergency alarms for patients to alert nurses.

4.2 Methodology for Adding Detected Gestures to Database along with Time Stamps

In our proposed architecture for hand detection module for adding detected gestures to Database along with Time Stamps:

- Database Setup: Before implementing the gesture detection system, we need to set up a suitable database to store the recognized gestures and their corresponding time stamps. The database should be designed to efficiently handle time-series data, allowing for easy retrieval and analysis.
- Database Schema: Define the database schema to accommodate the relevant information. The schema should include fields to store the following data:

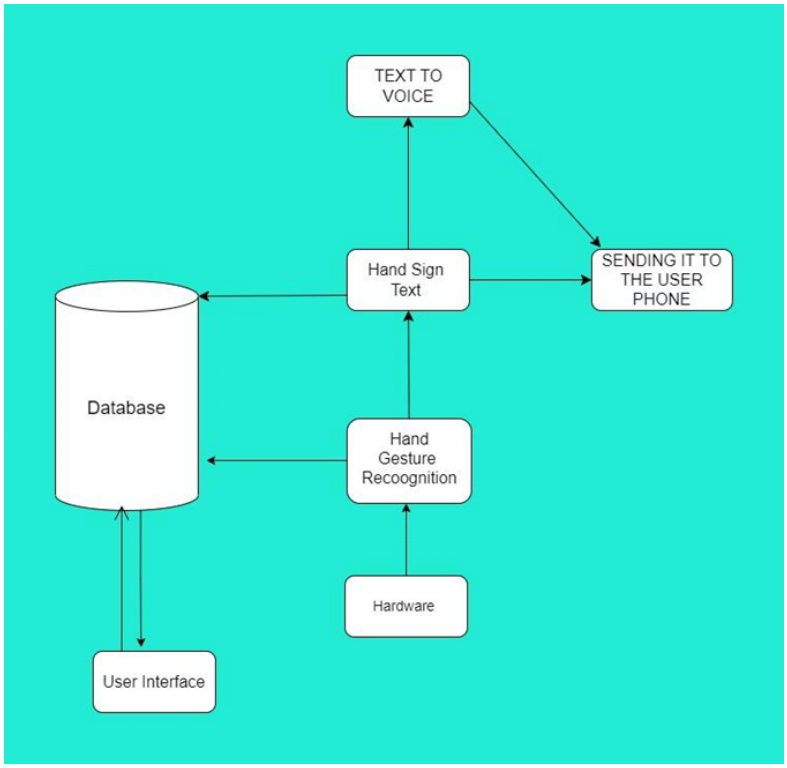


Figure 4.1: Architecture diagram

  - Gesture Name: The name or label of the recognized gesture.
  - Timestamp: The date and time when the gesture was detected.

- Gesture Detection and Recognition: Implement the hand gesture detection and recognition process as described in the previous methodology. Once a gesture is successfully recognized, it is assigned a unique Gesture ID and mapped to its corresponding Gesture Name (e.g., "hungry," "thirsty," "good," etc.).

- Timestamp Generation: When a gesture is detected and recognized, generate a timestamp corresponding to the exact moment when the gesture was identified. Use the system's internal clock or a real-time timestamp generator to ensure accurate time recording.

- Database Integration: Integrate the database with the gesture recognition system to allow seamless data storage. When a gesture is recognized, gather the relevant information (Gesture ID, Gesture Name, and Timestamp) and establish a connection to the database.

- Data Insertion: Implement a data insertion function to add the recognized gesture and its associated timestamp to the database. Use SQL or any other suitable database query language to perform the insertion operation. The function should take the detected gesture data as input and execute the query to insert the data into the designated table in the database.

- Error Handling: Incorporate error handling mechanisms in the data insertion process. Ensure that the system gracefully handles any exceptions or errors that might occur during the database insertion, such as connection failures or data conflicts.

- Database Management: Regularly maintain and manage the database to ensure optimal performance and data integrity. Consider implementing automated tasks, such as backups and data archiving, to prevent data loss and facilitate historical analysis.

- Data Retrieval and Analysis: To evaluate the system's performance and user behavior over time, implement data retrieval and analysis mechanisms. Utilize SQL queries or data analysis tools to extract relevant information from the database, such as the frequency of specific gestures, gesture usage patterns, and response times.

- Privacy and Security: If the system is used in sensitive contexts, such as healthcare, ensure that user privacy and data security are prioritized. Implement encryption protocols, access controls, and anonymization techniques to safeguard user information stored in the database.

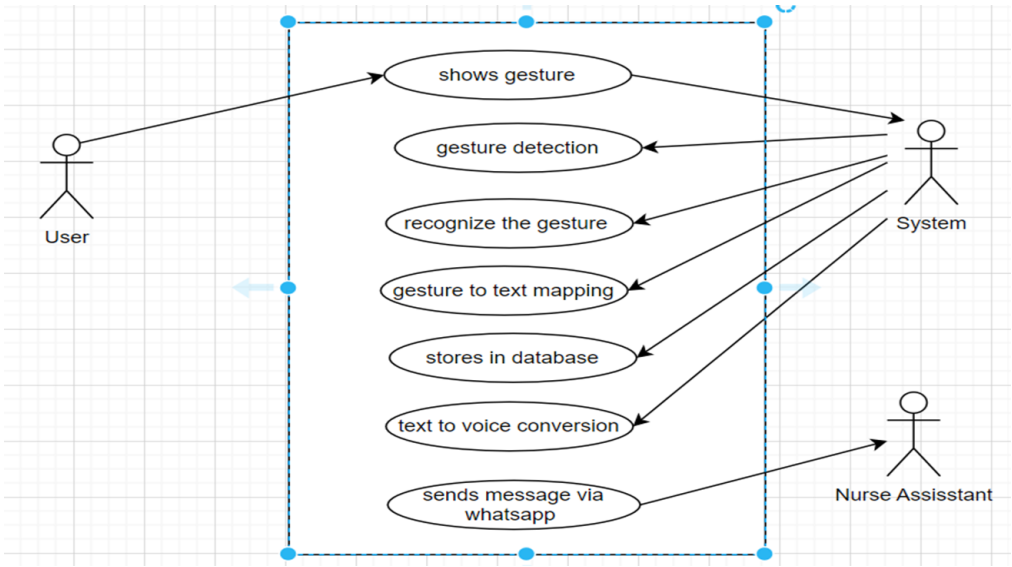
- Logging and Monitoring: Implement logging and monitoring functionalities to track system activities and potential issues. Log important events, such as database insertions, to facilitate debugging and performance optimization.

- Continuous Improvement: Continuously monitor and evaluate the system's performance, user feedback, and database usage. Use this information to make iterative improvements to the gesture recognition system and database management process. By following this methodology, the gesture recognition system can effectively store detected gestures in a database along with their respective time stamps, enabling valuable insights and analysis for research and evaluation purposes.

Chapter 5

System Design

5.1 Use case diagram



- Gesture show: user shows gesture to system when in need which is detected by system.
- Gesture detection Module: the gesture is detected by the system and its results is givn to next module which is detection.
- Gesture recognition Module: This module is responsible for capturing and processing hand gestures from input sources like cameras or depth sensors.
- Gesture-to-Text Mapping: Once the gestures are recognized, they are mapped to specific text or commands based on predefined gestures-to-text mappings.
- Text-to-Speech (TTS) Module: The mapped text or commands are then sent to the Text-to-Speech (TTS) module, which converts the text into natural-sounding voice output

- database module: To store gesture data in a database, we can use a relational database management system (RDBMS) like MySQL, which is commonly used for data storage.

- Sending Messages via WhatsApp Messages:

  To send voice outputs as WhatsApp messages, we can use the Twilio API, which provides APIs for sending WhatsApp messages programmatically. The pyttsx3 library can be used for text-to-speech conversion, and then the generated voice output can be sent as a WhatsApp message using the Twilio API.

5.2 Module-wise diagram

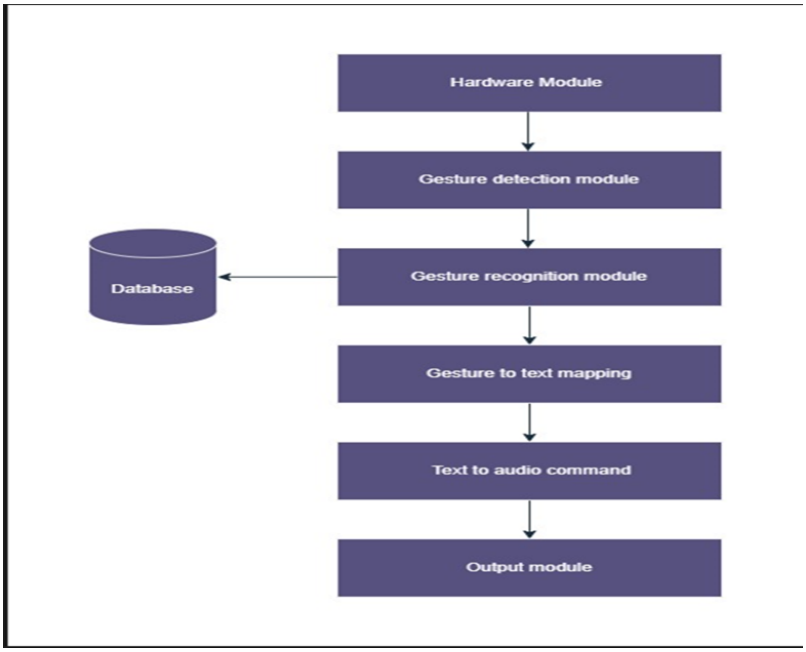


Figure 5.1: Module wise diagram

- Hardware Module: Basic input received from camera as hardware.

- Gesture detection module: Beginning from hand detection using hand detection neural network followed by convolution neural network,and hand landmark approach. The detected gestures are also sent to database for understanding the user's requirements and preferences.

- Gesture recognition module: Information from convolution neural network is passed on to feed forward neural network and using landmark mapping various gestures are recognized. Recognised gestures are added to database along with time stamp of use for further research and evaluation purposes.

- Gesture to text mapping : After the gestures are recognized they are named and mapped to various commands as texts.

- Message transfer: The recognised and mapped gestures are passed on to messaging applications such as whatsapp (*used for this project) to the assistants or nurse of the corresponding user in real time to notify them about the user's condition.

Chapter 6

System Implementation

6.1 Execute

6.1.1 Gesture Recognition

Training a Convolution Neural Network(CNN) from scratch takes a lot of data and also compute power. Instead we use transfer learning, where a model trained on similar data is fine-tuned as per our requirement. The keypoint-classification has 5 trained for hand gesture recognition as well as for face classification. We have used the keypoint-classification model as it is a smaller model and the prediction in real-time can work on local system with less GPU.

The keypoint-classification model was pre-trained on 5622 different hand gestures. The second to last layer has 4096 Dense Units, to which we append a 128 unit Dense layer, without the bias term, and remove the classification layer containing 2622 units. All the layers before the 128 Dense layer are frozen (trainable = False) and only the newly added dense layer needs to be trained.

Now for training this network, we use a logging key function. This function takes screenshots from user lively to add a new gesture and will write in the keypoint classifier csv file the name of the newly added gesture. The screenshots are taken from the numbers 0-9 by keeping on pressing for different sides,points and angles.This include for both the left and right hand based user convenience.

When a user presses the Run code button it redirects them to a detection Gui that is hosted on the same folder. The model is also saved on this local folder. On the website, the user has 2 options: add a new member or use the face recognition function.

If the user chooses to add a new member, they have to submit 10 images of the person who is being added, along with the name and relation (if necessary).

If the user chooses to use the face recognition function, the camera turns on and the name of all detected faces are shown, provided they were added beforehand.

6.1.2 Database storage

An additional feature to store the hand gestures recognized by the system in the mysql database as and when shown by the user to retrieve by the users when in need or to be searched based on date and time.

6.1.3 Emergency gesture

A gesture designed specifically for emergency. It helps the user to call the nurse in case of emergency with a specific emergency tune like an ambulance tune which is default and can be modified based on user requirement. It serves the need for people around to understand that user is in urgency.

6.1.4 Twilio feature for whatsapp

When the Run button is pressed it takes it to the gesture interface where the user shows the gesture which is recognized by the system and converted to text which is directly sent to Whatsapp via Twilio api for the nurse who receives it instantly and act based on it from anywhere remote. Twilio is a cloud communications platform that provides various APIs and services for sending messages, making phone calls, and handling other communication needs. With Twilio, developers can easily integrate communication functionalities into their applications and services.

One of Twilio's popular features is its API for WhatsApp, which allows developers to send and receive WhatsApp messages programmatically. Here's a short note on Twilio for sending WhatsApp messages:

- Twilio's API for WhatsApp enables developers to send and receive WhatsApp messages using Twilio's infrastructure.
- To use the API, developers need to sign up for a Twilio account and obtain their Account SID and Auth Token from the Twilio Console.

- A Twilio phone number that is enabled for WhatsApp is required to send and receive WhatsApp messages through the API. Developers can purchase a WhatsApp-enabled Twilio number from the Twilio Console.
- WhatsApp messages can be sent to recipients using their WhatsApp phone numbers in the format: whatsapp: +[country code][phone number].
- The API supports sending text, images, videos, documents, and other types of media as part of the WhatsApp message.
- Developers can use the Twilio Python library or other Twilio SDKs in various programming languages to interact with the Twilio API and send WhatsApp messages.

6.1.5 Gesture to text mapping

Gesture to text mapping is the process of converting hand gestures or body movements into text or written language. It is a part of gesture recognition systems that interprets the recognized gestures and translates them into textual information, allowing users to input text or communicate through gestures without the need for traditional keyboard typing.

Here's how gesture to text mapping typically works:

- Gesture Recognition: The process starts with capturing hand gestures or body movements using cameras or motion sensors. These gestures are then processed by the gesture recognition module, which identifies and classifies the gestures based on predefined patterns or machine learning algorithms.
- Gesture Preprocessing: Raw input data from the sensors might require preprocessing to enhance the quality of the gesture data, remove noise, and normalize the input for better accuracy during recognition.
- Feature Extraction: Relevant features from the preprocessed data are extracted to represent the gestures adequately. These features serve as inputs to the gesture recognition algorithms.

- Gesture Recognition Algorithms: The extracted features are used by machine learning algorithms or computer vision techniques to recognize specific gestures and map them to predefined gestures or commands.

- Gesture to Text Mapping: Once a gesture is recognized, the gesture to text mapping component comes into play. It takes the recognized gesture and converts it into corresponding text or language.

6.1.6 Text to Speech

When the Run button is pressed it takes it to the gesture interface where the user shows the gesture which is recognized by the system and converted to text. It is taken by the code and converted to a voice based signal and can be amplified so that the assistant can hear it from remote place and do the need based from remote area. The recognized commands are converted to audio or speech commands to give out as output. Text-to-Speech (TTS) synthesis can be implemented in Python using various libraries and APIs. One of the popular libraries for TTS in Python is the 'pyttsx3' library. 'pyttsx3' is a Python library for text-to-speech (TTS) synthesis, providing a simple and convenient way to convert written text into spoken audio. It offers cross-platform support and compatibility with Windows, macOS, and Linux, making it accessible to a wide range of users. One of the key advantages of 'pyttsx3' is its ability to work with multiple TTS engines, leveraging SAPI5 on Windows, NSSpeechSynthesizer on macOS, and espeak on Linux by default. With a straightforward API, 'pyttsx3' is easy to use, making it suitable for both beginners and experienced developers. Users can control the speaking rate and volume of the TTS output, enabling customization to suit individual preferences. The library also supports asynchronous operation, allowing users to multitask while the TTS engine processes and plays the speech in the background. Moreover, 'pyttsx3' is designed to be thread-safe, making it suitable for use in multi-threaded applications without concurrency issues.

Another significant feature is the ability to select different voices and languages for TTS output. Users can choose from available voices on their systems or install new ones, enabling a diverse range of synthesized speech options.

The library's extensibility is another advantage, enabling users to integrate custom or third-party TTS engines seamlessly. This adaptability ensures flexibility and the ability

to accommodate specific requirements.

Overall, ‘pyttsx3‘ is a versatile, user-friendly, and powerful text-to-speech synthesis library that empowers Python developers to incorporate spoken audio output into their applications with ease. Its wide range of features and compatibility with different platforms make it a popular choice for TTS implementations in various Python projects.

6.2 Search details

When the user presses the medicine search button in the menu, they are sent to the page to get the details of gestures on a particular date. Once the search button is pressed it displays the details of the user on that particular time and date. It is easy for users to check and review the details from the database based on need and time of user.

6.3 Help

This feature aims to guide user the gestures and its definition. Whenever the user forgets or needs help can refer this manual and retrieve it. It is easier for the admin to change it as and when based on flexibility and as and when time changes to add more gestures.

6.4 View all details

It is a feature or a manual for the user to learn the gestures and use them as and when needed while using the app. It makes the users aware of the set of gestures. It includes the details of time and gesture from the start of the present time.

6.5 Dataset

The Haarcascade dataset consisting of 2622 distinct celebrity images, is used for training the keypoint model used above. keypoint-classification consists of 8631 celebrity images for training, and 500 of them in the testing, each of them are distinct. The training set is 30GB. From that, a 2GB test set was used to train the last dense layer. For training and testing, the input images have to go through the same pre-processing that is defined by the keypoint-clasification model implemented by Oxford Group.

Chapter 7

Testing

7.1 Data Preparation

Prepare a separate testing dataset that is distinct from the training dataset used during the model training phase. This dataset should contain hand gesture images that the model has not seen before to simulate real-world scenarios.

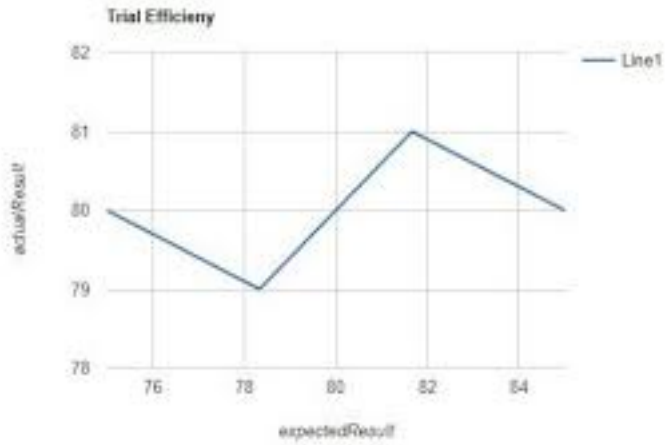


Figure 7.1: Actual Vs Expected results model

Performance Evaluation: Compare the predicted classes with the ground truth labels (the known correct labels) for the images in the testing dataset. Calculate various performance metrics such as accuracy, precision, recall, F1 score, and the confusion matrix to evaluate how well the model performs on the testing data.

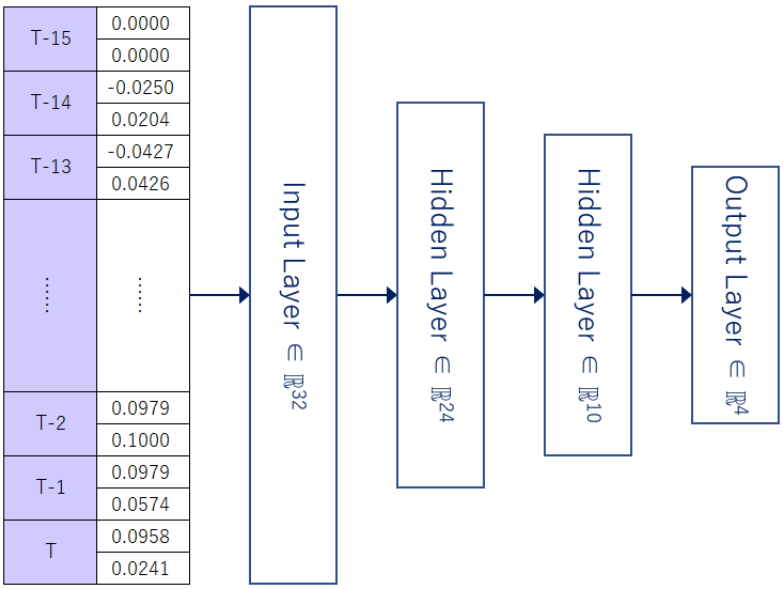


Figure 7.2: Keypoint classification model

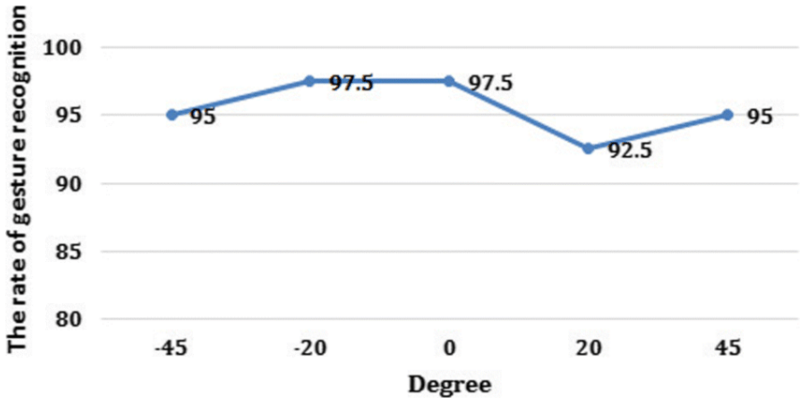


Figure 7.3: Rate of recognition Vs Degree model)

Visualizing Results: Optionally, you can visualize the model's performance by displaying some of the misclassified images and examining the patterns that might have caused misclassification. This analysis can provide insights into potential areas for improvement.

Chapter 8

Results

- Login page : In fig 8.1 each user is able to create an account and login to their respective account.
- Main Menu Interface: In fig 8.2 each user is able to access four options which include 'view details' , 'search' , 'execute' and ' help '.
- View all details: in fig 8.2 each the user is able to view all the requests asked by the patient or the user .This can be used to study the pattern of the patient.
- Search page : In fig 8.2 each user is able to enter the range of the date and able to access the specific requests asked during the particular time period .
- Whatsapp interface: In fig 8.5 each user shows the gesture which is recognized by the system and converted to text which is directly sent to Whatsapp via Twilio api for the nurse who receives it instantly and act based on it from anywhere remote.
- Help page: In fig 8.5 guides user the gestures and its definition. Whenever the user forgets or needs help can refer this manual and retrieve it.
- Gesture : In fig 8.7 and fig 8.8 are the examples of different gestures used .The first gesture is used to denote that the patient is hungry . The second gesture is used to denote that the patient is thirsty. there are other gestures denoting emergency , medicine , restroom etc.

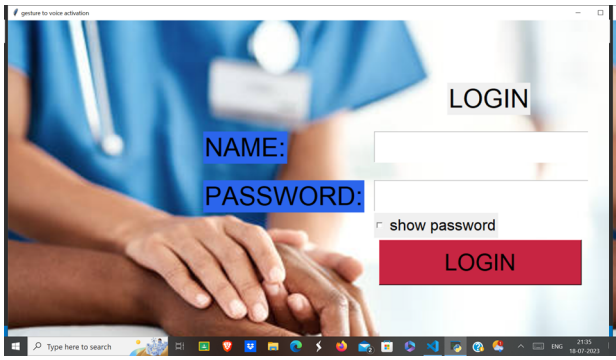


Figure 8.1: Login page

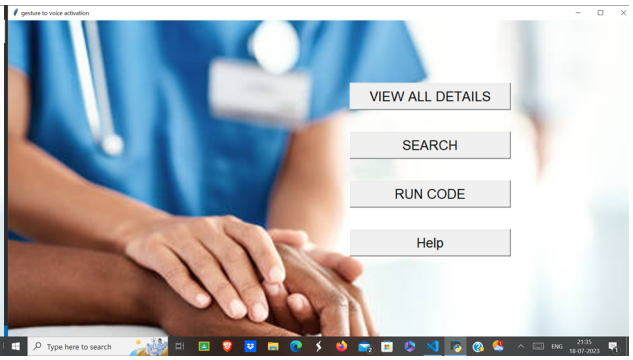


Figure 8.2: Main Menu interface

Figure 8.3: View all details

Figure 8.4: Search page based on date

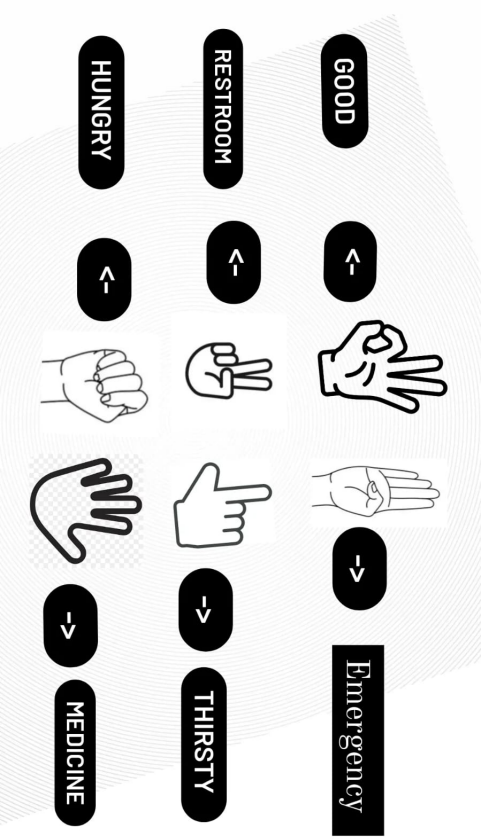


Figure 8.5: Help page

Figure 8.6: Whatsapp interface

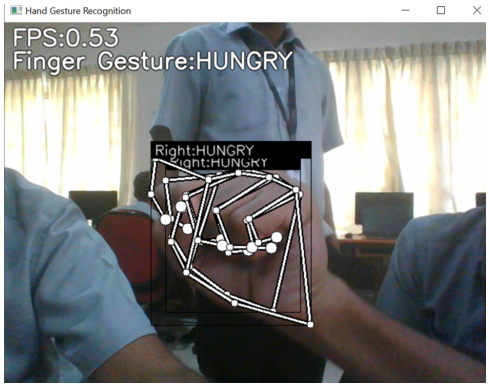


Figure 8.7: Hungry gesture

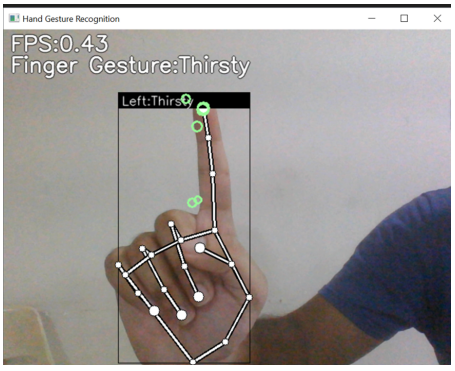


Figure 8.8: Thirsty gesture

Chapter 9

Risks and Challenges

1. Accuracy and Robustness: One of the primary challenges in hand gesture to voice conversion is achieving high accuracy and robustness in recognizing and interpreting hand gestures.

2. Limited Gesture Vocabulary: The system's effectiveness may be limited by the predefined set of gestures and corresponding voice outputs.

3. Individual Variations: Users have different hand sizes, shapes, and mobility capabilities, which can result in variations in hand gestures.

4. Latency: Reducing the time between making a gesture and receiving the corresponding voice output is crucial for a smooth user experience. Minimizing latency requires efficient algorithms and processing power.

5. Environmental Factors: The environment in which the gesture to voice recognition system operates can impact its performance. Background noise, varying lighting conditions, and distractions might hinder accurate gesture detection.

Chapter 10

Conclusion

This software is meant to help people who face difficulties in speaking and also elderly people The main features of the app involve : ∘ Hand gesture recognition ∘ Speech to text conversion ∘ Text to voice conversion ∘ Message sending It uses a hybrid approach that combines deep learning techniques with sensor-based systems offers a promising solution for accurate and robust hand gesture recognition ensuring real-time performance which is crucial for an effective user experience. The system is also designed to have a user-friendly interface, enabling users to learn and perform gestures easily.

References

[1] Okan Kopuklu, Ahmet Gunduz, Neslihan Kose, Gerhard Rigoll ,18 OCT 2019 "Real-time Hand Gesture Detection and Classification UsingConvolutional Neural Networks" IEEE International Conference on Automatic Face and Gesture Recognition (FG 2019)

[2] G. R. S. Murthy and R. S. Jadon "A Review of Vision Based Hand Gestures Recognition" International Journal of Information Technology and Knowledge Management, July-December 2009, Volume 2, No. 2, pp. 405-410

[3] Viraj Shinde, Tushar Bacchav, Jitendra Pawar, Mangesh Sanap "Hand Gesture Recognition System Using Camera" International Journal of Engineering Research and Technology (IJERT) Vol. 3 Issue 1, January - 2014 IJERT ISSN: 2278-0181

[4] M S Srividya, Anala M R , "Research trends in Hand Gesture Recognition techniques"International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878 (Online), Volume-8 Issue-6, March 2020

[5] Coşkun, M., Uçar, A., Yildirim, Ö. and Demir, Y., 2019, November. Face recognition based on convolutional neural network. In 2019 International Conference on Modern Electrical and Energy Systems (MEES) (pp. 376-379). IEEE.

[6] Lo, W.W., Yang, X. and Wang, Y., 2019, June. An xception convolutional neural network for malware classification with transfer learning. In 2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS) (pp. 1-5). IEEE.

[7] Yoichi Sato, Institute of Industrial Science, the University of Tokyo, Makiko Saito Hideki Koike, Graduate School of Information Systems, University of Electro- Communications, Tokyo Real-Time Input of 3D Pose and Gestures of a User's Hand and Its Applications for HCI

[8] Manuel Cabido Lopes Jos´e Santos-Victor Instituto de Sistemas e Rob´otica, Institution Superior T´ecnico, Lisbon, Portugal in IROS Workshop on Robot Programming by demonstration, Las Vegas, SA, Oct 31st, 2003 Motor Representations for Hand Gesture Recognition and Imitation

[9] Didier Stricker, Didier Stricker, June 15th 2006 Hand gesture recognition gradient orientation histograms and eigenvectors methods

[10] Steven Daniel Lovell, February 2005, A System for Real-Time Gesture Recognition and Classification of Coordinated Motion

[11] Klimis Symeonidis, Submitted to the School of Electronic and Electrical Engineering On August 23,2000 Hand Gesture Recognition Using Neural Networks.

Appendix A: Base Paper

applied sciences

MDPI

Article

Real-Time Hand Gesture Recognition Based on Deep Learning YOLOv3 Model

Abdullah Mujahid [1], Mazhar Javed Awan [2], Awais Yasin [3], Mazin Abed Mohammed [4], Robertas Damaševičius [5,*], Rytis Maskeliūnas [6] and Karrar Hameed Abdulkareem [7]

1 Department of Computer Science, University of Management and Technology, Lahore 54770, Pakistan; abdullahmujahidali1@gmail.com
2 Department of Software Engineering, University of Management and Technology, Lahore 54770, Pakistan; mazhar.awan@umt.edu.pk
3 Department of Computer Engineering, National University of Technology, Islamabad 44000, Pakistan; awaisyasin@nutech.edu.pk
4 Information Systems Department, College of Computer Science and Information Technology, University of Anbar, Anbar 31001, Iraq; mazinalshujeary@uoanbar.edu.iq
5 Faculty of Applied Mathematics, Silesian University of Technology, 44-100 Gliwice, Poland
6 Department of Applied Informatics, Vytautas Magnus University, 44404 Kaunas, Lithuania; rytis.maskeliunas@vdu.lt
7 College of Agriculture, Al-Muthanna University, Samawah 66001, Iraq; Khak9784@mu.edu.iq
* Correspondence: robertas.damasevicius@polsl.pl

Abstract: Using gestures can help people with certain disabilities in communicating with other people. This paper proposes a lightweight model based on YOLO (You Only Look Once) v3 and DarkNet-53 convolutional neural networks for gesture recognition without additional preprocessing, image filtering, and enhancement of images. The proposed model achieved high accuracy even in a complex environment, and it successfully detected gestures even in low-resolution picture mode. The proposed model was evaluated on a labeled dataset of hand gestures in both Pascal VOC and YOLO format. We achieved better results by extracting features from the hand and recognized hand gestures of our proposed YOLOv3 based model with accuracy, precision, recall, and an F-1 score of 97.68, 94.88, 98.66, and 96.70%, respectively. Further, we compared our model with Single Shot Detector (SSD) and Visual Geometry Group (VGG16), which achieved an accuracy between 82 and 85%. The trained model can be used for real-time detection, both for static hand images and dynamic gestures recorded on a video.

Keywords: convolutional neural network; hand gesture; digital image processing; YOLOv3; artificial intelligence

Citation: Mujahid, A.; Awan, M.J.; Yasin, A.; Mohammed, M.A.; Damaševičius, R.; Maskeliūnas, R.; Abdulkareem, K.H. Real-Time Hand Gesture Recognition Based on Deep Learning YOLOv3 Model. *Appl. Sci.* **2021**, *11*, 4164. https://doi.org/10.3390/app11094164

Academic Editor: Athanasios Nikolaidis

Received: 3 April 2021
Accepted: 29 April 2021
Published: 2 May 2021

1. Introduction

The interaction between humans and computers has increased widely, while the domain is witnessing continuous development, with new methods derived and techniques discovered. Hand gesture recognition is one of the most advanced domains in which computer vision and artificial intelligence has helped to improve communication with deaf people but also to support gesture-based signaling systems [1,2]. Subdomains of hand gesture recognition include sign language recognition [3–5], recognition of special signal language used in sports [6], human action recognition [7], pose and posture detection [8,9], physical exercise monitoring [10], and controlling smart home/assisted living applications with hand gesture recognition [11].

Over the years, computer scientists have used different computation algorithms and methods to help solve our problems while easing our lives [12]. The use of hand gestures in different software applications has contributed towards improving computer and human interaction [13]. The progress of the gesture recognition systems plays a vital role in the

development of computer and human interaction, and the use of hand gestures in various domains is growing more frequent. The application of the use of hand gestures can now be seen in games [14], virtual reality [15,16] and augmented reality [17], assisted living [18,19], cognitive development assessment [20], etc. The recent development of hand gesture recognition in different sectors has grabbed the attention of industry, too, for human-robot interaction in manufacturing [21,22], and control of autonomous cars [23].

The main aim of this real-time hand gesture recognition application is to classify and recognize the gestures. Hand recognition is a technique in which we use different algorithms and concepts of various techniques, such as image processing and neural networks, to understand the movement of a hand [24]. In general, there are countless applications of hand gesture recognition. For example, for deaf people who cannot hear, we can communicate with their familiar sign language.

There are many object detection algorithms that help to detect and determine what the gesture is that each algorithm targets. This paper explores a few algorithms and detects which algorithm is better than the others, providing better accuracy with fast and responsive results. To achieve this detection, You Only Look Once (YOLO) v3 and Single Shot Detector (SSD) algorithms were used to evaluate the structure and mechanism deduction of hand gesture recognition.

YOLO is a convolutional neural network algorithm, which is highly efficient and works tremendously well for real-time object detection [25,26]. A neural network not only helps in feature extraction, but it can also help us understand the meaning of gesture, and help to detect an object of interest. A similar approach was adopted in [27], but the main difference is that in [27] it was done through Light YOLO, which is completely different from YOLOv3. Light YOLO is preferred for applications built on RaspberryPi. It achieves a good frame per second (fps) rate as compared to YOLOv3. However, the accuracy of YOLOv3 is 30% better when compared to Light YOLO, and as the application is not focused on the Internet-of-Things (IoT) product, YOLOv3 is preferred.

The main contributions of this study are summarized as follows:

- A lightweight proposed model where there is no need to apply as much preprocessing which involves filtering, enhancement of images, etc.;
- A labeled dataset in both Pascal VOC and YOLO format;
- This is the first gesture recognition model that is dedicated to the mentioned gestures using YOLOv3. We use YOLOv3 as it is faster, stronger, and more reliable compared to other deep learning models. By using YOLOv3, our hand gesture recognition system has achieved a high accuracy even in a complex environment, and it successfully detected gestures even in low-resolution picture mode;
- The trained model can be used for real-time detection, it can be used for static hand images, and it also can detect gestures from video feed.

The organization of our study is as follows: Section 2 presents the related work and reviews the latest studies on hand gesture recognition. The materials and methods that were used in this study are in Section 3. Section 4 presents the results of the Real-Time Hand Gesture Recognition Based on Deep Learning YOLOv3 Model. The discussion of the results of the proposed Real-Time Hand Gesture Recognition Based on Deep Learning YOLOv3 Model are discussed in Section 5. Finally, Section 6 concludes the study directions and future works.

2. Related Work

There are various studies on hand gesture recognition as the area is widely expanding, and there are multiple implementations involving both machine learning and deep learning methods aiming to recognize a gesture that is intonated by a human hand. Further, some papers are reviewed to understand the mechanism of the hand gesture recognition technique.

The study [28] demonstrated that with insignificant computational cost, one of the normal and well-known designs, CNN, accomplished higher paces of perceiving components effectively. The proposed strategy focused only on instances of gestures present

in static images, without hand location, and followed the instances of hand impediment with 24 gestures, utilizing a backpropagation algorithm and segmentation algorithm for preparing the multi-layer propagation, and for the backpropagation calculation having its influence, sorting out the blunder proliferated on the contrary request [29]. Moreover, it utilized a convolutional neural organization, and sifting incorporated a distinctive division of image division and recognition.

Among these techniques and methods there is a popular method which is used by several other detection applications, and that is Hidden Markov Models (HMM). There are different detection variants that are often used by an application, which we have come across, and the application this paper refers to checks and works with all of it by learning and looking towards other papers. This application deals with all three mediums: image, video, and webcam [30]. Going through a general paper written by Francois, in which he refers to an application that detects posture, the detection is through video and uses the HMM. The process that the paper of Francois and other researchers worked on is related to this application, which this paper refers to first by extracting the information from either image, video, or real-time detection using webcams. These features are detected by the three methods mentioned, but the main concern of all the methods, whether they are CNN-related, RNN-related, or using any other technique, is that all of them use fitting techniques, and these techniques refer to the bounding box that this paper discussed. The box represents the information that is detected, from which they gain a confidence value, and the value that is the highest is the output of what image is being displayed. Besides this, all other information varies depending upon the method that others are using; despite that, some other devices and techniques that are related to segmentation, general localization, and even fusion of other different partials help achieve the tasks of detecting and recognizing.

Nyirarugira et al. [31] proposed Social Touch Gesture Recognition using CNN. He utilized various calculations such as Random Forest (RF), boosting algorithms, and the Decision Tree algorithm to recognize gestures, utilizing 600 arrangements of palmar images with 30 examples utilizing convolutional neural organization, and finding an ideal method on the premise of the framework of an 8×8 network. The casing length is variable anyway so the outcome decides the ideal casing length. It is performed using a dataset as of late assessed with different subjects that perform changing social gestures [32]. A system that gathers contact gestures in a practically constant manner using a deep neural organization is favorable to present. The results showed that their strategy performed better when differentiated, and the previous work was reliant on leave-one-subject-out cross-endorsement for the CoST dataset. The proposed approach presents two points of interest differentiated from those in the current writing, acquiring an accuracy of 66.2% when perceiving social gestures.

In the study of Multiscale CNNs for hand detection, excited by the headway of article recognition in the field of PC vision, numerous techniques have been proposed for hand-distinguishing proof in the latest decade [33]. The most untroublesome procedure relies upon the recognition of skin tone, which works on hands, faces, and arms, yet moreover has issues because of the affectability for brilliant changes. The contributing components of this multifaceted nature fuse substantial hindrance, low objectivity, fluctuating illumination conditions, various hand gestures, and the incredible coordinated efforts among hands and dissents or various hands. They further presented a Multiscale Fast R-CNN approach to manage to correctly recognize human hands in unconstrained pictures. By merging staggered convolutional features, the CNN model can achieve favored results over the standard VGG16 model, accomplishing practically 85% of 5500 images for the testing and 5500 for the preparing set.

Saqib et al. [34] used a CNN model augmented by edit distance for the recognition of static and dynamic gestures of Pakistani sign language, and achieved 90.79% accuracy. Al-Hammadi et al. [35] proposed a 3DCNN model to learn region-based spatiotemporal features for hand gestures. The fusion techniques to globalize the local features learned

by the 3DCNN model were used to improve the performance. The approach obtained recognition rates of 98.12, 100, and 76.67% on three color video gesture datasets.

Do et al. [36] proposed a multi-level feature LSTM with Conv1D, the Conv2D pyramid, and the LSTM block. The proposed method exploited skeletal point-cloud features from skeletal data, as well as depth shape features from the hand component segmentation model. The method achieved accuracies of 96.07 and 94.40% on the Dynamic Hand Gesture Recognition (DHG) dataset with 14 and 28 classes, respectively. The study extracted diversity of dynamic hand gestures from 14 depth and 28 skeletal data through the LSTM model with two pyramid convolutional blocks. The accuracy of 18 classes was 94.40%. Elboushaki et al. [37] learned high-level gesture representations by using Convolutional Residual Networks (ResNets) for learning the spatiotemporal features from color images, and Convolutional Long Short-Term Memory Networks (ConvLSTM) to capture the temporal dependencies between them. A two-stream architecture based on 2D-ResNets was then adopted to extract deep features from gesture representations.

Peng et al. [38] combined a feature fusion network with a ConvLSTM network to extract spatiotemporal feature information from local, global, and deep aspects. Local feature information was acquired from videos by 3D residual network, while the ConvLSTM network learned the global spatiotemporal information of a dynamic gesture. The proposed approach obtained 95.59% accuracy on the Jester dataset, and 99.65% accuracy on the SKIG (Sheffield Kinect Gesture) dataset. Tan et al. [39] proposed an enhanced, densely connected convolutional neural network (EDenseNet) for hand gesture recognition. The method achieved 99.64% average accuracy on three hand gesture datasets. Tran et al. [40] suggested a 3D convolution neural network (3DCNN) that could extract fingertip locations and recognize hand gestures in real-time. The 3DCNN model achieved 92.6% accuracy on a dataset of videos with seven hand gestures.

Rahim et al. [41] analyzed the translation of the gesture of a sign word into text. The authors of this paper performed the skinMask segmentation to extract features along the CNN. Having a dataset of 11 gestures from a single hand and 9 from double hands, the support vector machine (SVM) was applied to classify the gestures of the signs with an accuracy of 97.28%. Mambou et al. [42] analyzed hand gestures associated with sexual assault from indoor and outdoor scenes at night. The gesture recognition system was implemented with the combination of the YOLO CNN architecture, which extracted hand gestures, and a classification stage of bounding box images, which lastly generated the assault alert. Overall, the network model was not lightweight and had a lower accuracy.

Ashiquzzaman et al. [43] proposed a compact spatial pyramid pooling (SPP) a CNN model for decoding gestures or finger-spelling from videos. The model used 65% fewer parameters than traditional classifiers and worked 3× faster than classical models. Benitez-Garcia et al. [44] employed a lightweight semantic segmentation FASSD-Net network, which was improved over Temporal Segment Networks (TSN) and Temporal Shift Modules (TSM). They demonstrated the efficiency of the proposal on a dataset of thirteen gestures focused on interaction with touchless screens in real-time.

Summarizing, the biggest challenge faced by the researchers is designing a robust hand gesture recognition framework that overcomes the most typical problems with fewer limitations and gives an accurate and reliable result. Real-time processing of hand gestures also has some limitations, such as illumination variation, background problems, distance range, and multi-gesture problems. There are approaches to hand gesture recognition that use non-machine-learning algorithms, but there is a problem in that the accuracy varies, and in different environments such as light, it overlaps one gesture with another, which makes the approach less flexible and unable to adapt independently, when compared to the machine-learning approach. Therefore, the machine-learning approach was used for developing the system.

3. Material and Methods

This section is dedicated to the materials and the methods that were used in this study to achieve the gesture recognition that this paper aimed for. Section 3.1 explains the dataset and all the information related to the material. Section 3.2 deals with the algorithm and the methods that we used to solve the problem.

3.1. Dataset

Our dataset consisted of 216 images. These images were further classified into 5 different sets. Each set held an average of 42 images which were labeled using the YOLO labeling format. The dataset was labeled using a labeling tool, which was an open-source tool used to label custom datasets. We used the YOLO format, which labels data into text file format and holds information such as the class ID and class to which it belongs. Our classes started from 0 to 4, where 0 class ID is labeled as 1 and 4 class ID is labeled as 5. There were a total of 5 sets that our application detected, which were finger-pointing positions of 1, 2, 3, 4, and 5. Figure 1 displays the hand gestures in our collected dataset.

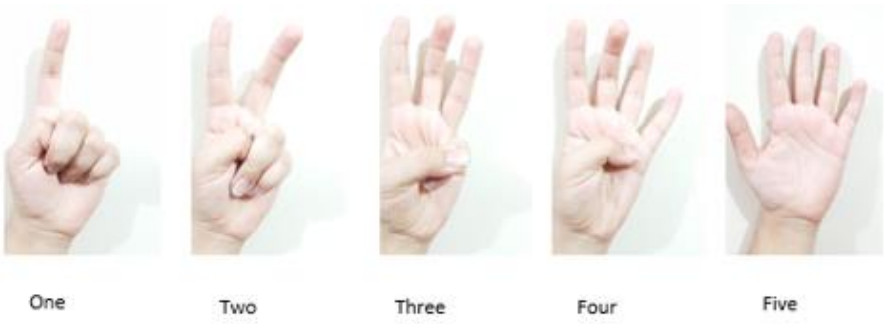


Figure 1. Hand gesture set to be detected.

3.2. Data Preprocessing

Data preprocessing is an important part of the before training and testing phase. Using a YOLO configuration with a total of 200+ images of the dataset, where 30 were used for the affine transformation by increasing 2-fold, each image was duplicated for reading and training, both left and right hand, by flipping it horizontally and sometimes taking the respective image of those hands for making the set more accurate. Furthermore, an additional 15 images were taken and labeled for the testing set. The data preprocessing step is important before moving towards post-processing, because we have to look at what type of data we have collected and which part of it will be useful for the purpose of training, testing, and for obtaining better accuracy. Table 1 shows the instances of five classes with the features of the YOLO-labeled data.

Table 1. Summary of data obtained from YOLO-labeled data file.

Class ID	X-Cord	Y-Cord	Width	Height
0	0.531771	0.490234	0.571875	0.794531
1	0.498437	0.533203	0.571875	0.905469
2	0.523438	0.579297	0.613542	0.819531
3	0.526563	0.564453	0.473958	0.819531
4	0.498611	0.587891	0.977778	0.792969

The above example is the representation of how these files will look when we label our dataset for training it on the desired model. Each line contains 5 different attributes, and all these attributes have their importance. Looking at the left first, we have class ID, followed by the next two, which are the labeled box co-ordinates of a gesture with the x-axis and y-axis values, followed by the width and height of that annotated image.

Figure 2 represents the correlation heat map of the YOLO-labeled dataset. Here, we can see the diversity of values among different labels, explaining the concentration and depth of our images represented in the form of a multi-dimensional matrix.

Figure 2. Heat map of the YOLO-labeled dataset.

3.3. Proposed Method

To understand the method which we are proposing, we need to look at the diagram presented in Figure 3 to understand better, generally, what and how our application is detecting objects. It is a kind of a general overview of the application that we have developed. The training process first requires collecting the dataset, and after that the next step is to label it, so we use YOLO annotation to label our data, which gives us some values that are later explained in the model process. After that, when the data is labeled, we then feed it to the DarkNet-53 model, which is trained according to our defined configuration. The image is captured through the camera that can be an integrated (primary) camera or it can be any external (secondary) camera. Other than that, the application can also detect gestures from a video input as well.

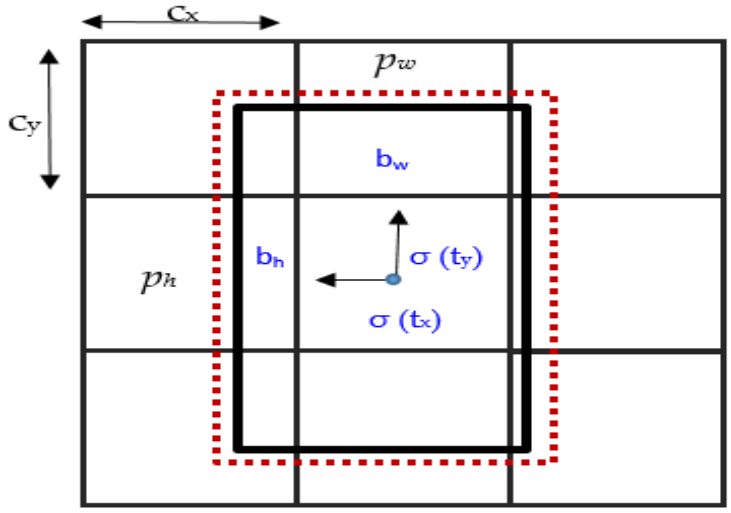


Figure 3. The bounding box highlighting positions of the values for further classification.

After capturing real-time objects with the help of the OpenCV [45] module, which is an open-source computer vision library, we can capture images and after that, we send them frame by frame from the real-time objects. Because of incorrect filtering, our dataset of collected images currently has a variable number of images. These images are labeled according to the classes we have stored for our YOLO algorithm, so we have successfully attained the coordinate and the class for our image set. After that, we can now set towards the training section. We then pass this set to our training algorithm, which is a deep neural network model YOLO.

Further, we discuss the methodology how YOLO deals with the network desired output which is achieved by using a formula which takes different co-ordinates, such as pw, ph, tx, ty, tw, th, cx, and cy. These are the variables which we use for the bounding box dimensions. Obtaining the values of the boundary box (x-axis, y-axis, height, and width) is described by Equation (1).

$$\begin{aligned} b_x &= \alpha\left(\mathrm{t}_x\right) + \mathrm{Cx} \\ b_y &= \alpha\left(\mathrm{t}_y\right) + \mathrm{Cy} \\ b_w &= p_w e^t w \\ b_h &= phe^t h \end{aligned} \tag{1}$$

where bx, by, bw, and bh are the box prediction components, x and y refer to the center co-ordinates, and w and h refer to the height and width of the bounding box. Equation (1) used in the YOLOv3 algorithm shows how it extracts the values from the image in the bounding box, and below is the diagram of how these values are extracted from the bounding box.

From Figure 3, we can understand how each value of the bounding box from the algorithm provides us with the co-ordinates of the center, x and y. From the prediction here the next important thing comes, which is the sigmoid function, which we have already discussed above, which filters out data except for the main part which is going to be recognized.

From Figure 4, we can understand the backend of the algorithm. When we pass an input, it first comes into the input layer, and after that it is further rendered and passes into the hidden layers, that are several in number and size, and are interconnected convolutions. These convolutional layers determine a special value, which is the value of confidence. In our case, if the value of the confidence threshold is greater than 0.5, then we assume that the application has successfully determined what object it has encountered.

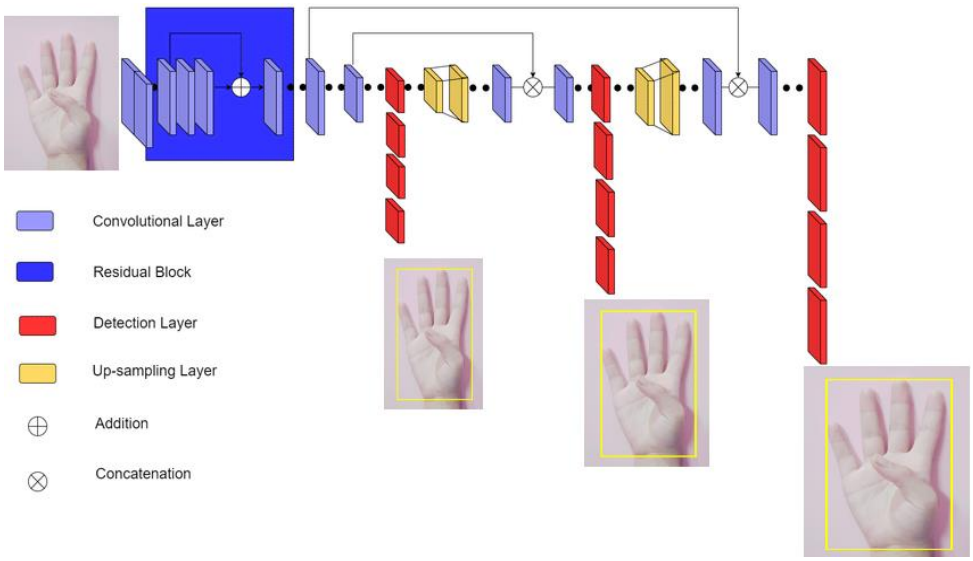


Figure 4. Architecture of the neural network for hand gesture recognition.

3.4. Implementation

Figure 5 explains how the YOLO algorithm plays its part when it acquires the image with the help of the OpenCV module. The image is then passed to the YOLO network, which then further identifies the required target. After doing that it sends it forward to the feature map prediction block, where it further extracts the information which is required to identify the gesture, then, after predicting it, sends it to the decoding part, where the output predicted is mapped onto the image and then displayed, as shown in Figure 5.

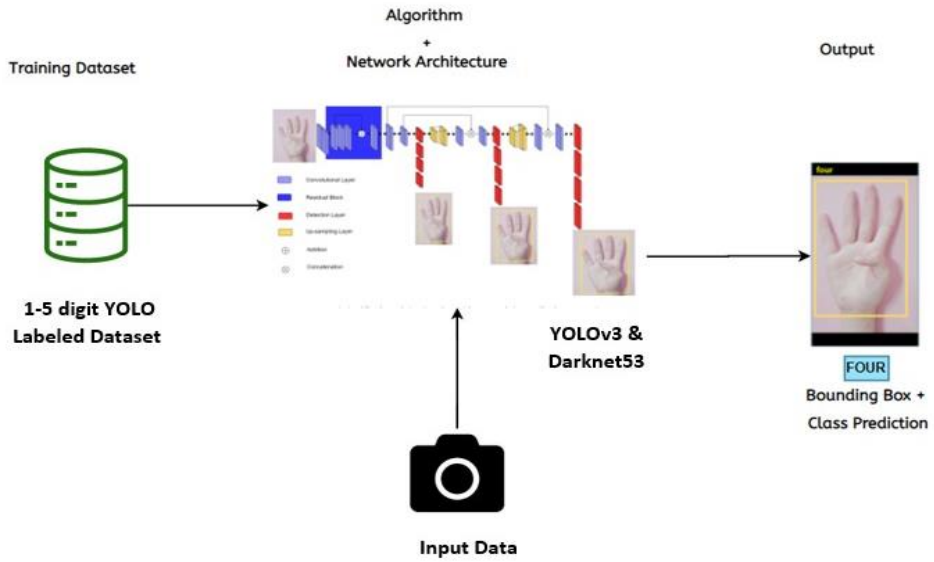


Figure 5. In-depth architectural flow of how the application deals with the recognition and detection of the input image with a gesture using YOLOv3 network.

We changed the configuration of YOLO and defined the activation according to the stride and the pad. For the YOLOv3 model, we set the mask to 0.5. The learning rate was set to 0.001, and the value of jitter was set to 0.3.

For the exploratory study of the best alternatives to implement the gesture recognition system, we use some other models to check which algorithm works best for gesture detection which will be discussed in the next section.

The developed neural network model is summarized in Table 2. Our input layer is just a typical CNN, which has convolutional layers, and other than that, it has a special layer, which is a max-pooling layer, and a very simple layer, which is the output layer. The general architecture of the application, which includes both the training and testing set, can be seen in Figure 6.

Table 2. Model Summary.

Layer (Type)	Output Shape	Parameters
Conv2d_133 (Conv2D)	(None, 126, 254, 64)	1792
Max_pooling_132 (Max Pooling)	(None, 63, 127, 64)	0
Conv2d_134 (Conv2D)	(None, 61, 125, 64)	36928
Max_pooling_133 (Max Pooling)	(None, 20, 41, 64, 0)	0
Conv2d_135 (Conv2D)	(None, 18, 39, 64)	36928
Max_pooling_134 (Max Pooling)	(None, 6, 13, 64)	0
Conv2d_136 (Conv2D)	(None, 4, 11, 64)	36928
Max_pooling_135 (Max Pooling)	(None, 1, 3, 64)	0
Flatten_33 (Flatten)	(None, 192)	0
Dense_151 (Dense)	(None, 128)	24704
Dropout_36 (Dropout)	(None, 128)	0
Dense_152 (Dense)	(None, 64)	8256
Dense_153 (Dense)	(None, 32)	2080
Dense_154 (Dense)	(None, 8)	264

Total params: 147,880. Trainable params: 147,880. Non-trainable params: 0.

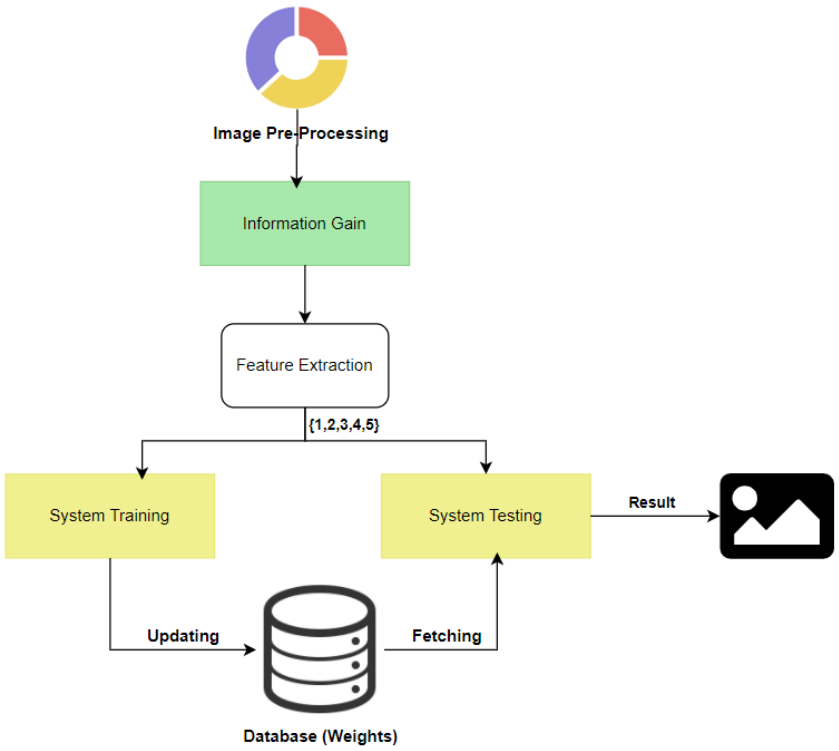


Figure 6. General architecture of the developed hand gesture detection system.

4. Results

In this section, we present, discuss, and evaluate our results.

4.1. Environmental Setup

To carry out this experiment Python 3.7 was used to train the algorithm on the personal computer with local 4GB GPU. Other important parameters are presented in Table 3. The training of the neural network model on our dataset took more than 26 h on a GPU of 24 GB.

Table 3. Parameters and values for environment setup.

Hyper Parameter	Value
Learning Rate	0.001
Epochs	30
No. of Classes	5
Algorithm	DarkNet-53 and YOLOv3
Optimizer	Adam
Activation	Linear, Leaky
Filter Size	[64,128,256,512,1024]
Mask	0–8
Decay	0.0005

4.2. Results and Performance of YOLOv3

Figure 7 shows the output of the developed application that performs real-time hand gesture recognition. As you can see, the bounding box is a little bit large, which is because it was left intentionally as 416 × 416 for covering the maximum part, then scattered, and all unnecessary information was removed from the image so we could obtain a larger coverage area. This also helps in the zoom case when the object is too large, as it will cleverly identify which gesture is being represented, and it then returns the class ID with the best match.

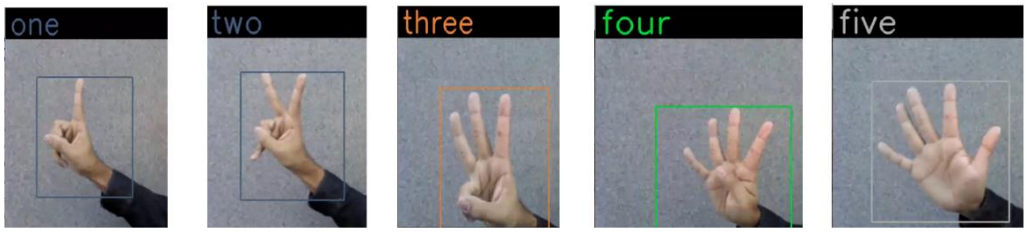


Figure 7. Gesture detected using the proposed YOLOv3-based methodology.

Figure 8 shows the training performance of the proposed deep learning model. The results are 98% correct, as measured by experiments. The experiment is in real time because we trained our model with YOLO and Pascal VOC [46] configurations and tested it with live images. We added the YOLO-annotated labels into a CSV file and re-ran training tests to confirm the accuracy of the model by plotting the curve, as we can find the accuracy of an individual gestation in the real-time experiment. The fault was seen in the transitions between one gesture to another, because the application is in real-time, so the system can detect dynamic objects too, as proven by our experiments.

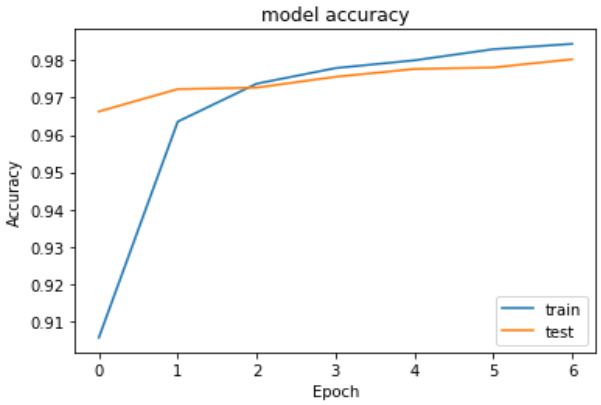


Figure 8. The performance of the proposed model.

We used several different algorithms to test which was the best method for the gesture recognition application. We compared YOLOv3 [47] with other deep learning models

(VGG16 [48] and SSD [49]). In the end, YOLOv3 produced the best results and was chosen, with an accuracy of 97.68% during training and an outstanding 96.2% during testing. DarkNet-53 was used to train the dataset, and for the detection, YOLOv3 was used. As we know, in real-time experiments it is not quite possible to replicate the gestures exactly, so in order to determine the accuracy of the model, so we generated a CSV file of a few YOLO-annotated labels and re-ran the code with some modifications for the accuracy curve.

Table 4 explains the accuracy of the individual models, where the stochastic gradient descent gave the lowest value because the real-time motion stochastic could not identify moving objects correctly, compared to the other algorithms. We evaluated the performance of deep learning models using the precision, recall, F-1 score, and accuracy measures, achieving an accuracy of 97.68% for YOLOv3.

Table 4. Comparison of the accuracy results for the deep learning models used for hand gesture recognition. Best results are shown in bold.

Models	Learning Rate	Images	Precision (%)	Recall (%)	F-1 Score (%)	Accuracy (%)
VGG16	0.001	216	93.45	87.45	90.3	85.68
SGD	0.1	216	70.95	98.37	82.4	77.98
SSD	0.0001	216	91.25	84.30	87.6	82.00
YOLOv3	0.001	216	94.88	98.68	96.7	97.68

Table 5 shows the comparison of the state-of-the-art works of Chen et al. [28], Nyirarugira et al. [31], Albawi et al. [32], Fong et al. [50], Yan et al. [51], and Ren et al. [52] with our proposed model, which produced better results with higher accuracy.

Table 5. Comparison of the state-of-the-art works with our proposed model.

Reference	Model	Dataset	Accuracy (%)
Chen et al. [28]	Labeling Algorithm Producing Palm Mask	1300 images	93%
Nyirarugira et al. [31]	Particle Swarm Movement (PSO), Longest Common Subsequence (LCS)	Gesture vocabulary	86%
Albawi et al. [32]	Random Forest (RF) and Boosting Algorithms, Decision Tree Algorithm	7805 gestures frames	63%
Fong et al. [50]	Model Induction Algorithm, K-star Algorithm, Updated Naïve Bayes Algorithm, Decision Tree Algorithm	50 different attributes total of 9000 data instance 7 videos	76%
Yan et al. [51]	AdaBoost Algorithm, SAMME Algorithm, SGD Algorithm, Edgebox Algorithm	5500 images for the testing and 5500 for the training set	81.25%
Ren et al. [52]	FEMD Algorithm, Finger Detection Algorithm, Skeleton-Based Matching	1000 cases	93.20%
Proposed model	YOLOv3	216 images (Train) 15 images (Test)	97.68%

5. Discussion

Real-time hand gesture recognition based on deep learning models has critical roles in many applications due to being one of the most advanced domains, in which the computer vision and artificial intelligence methods have helped to improve communication with deaf people, but also to support the gesture-based signaling systems]. In this study, we experimented with hand gesture recognition using YOLOv3 and DarkNet-53 deep learning network models [47]. The dataset, which we used, was collected, labeled, and trained by

ourselves. We compared the performance of YOLOv3 with several other state-of-the-art algorithms, which can be seen in Table 5. The achieved results were good, as YOLOv3 achieved better results when compared to other state-of-the art algorithms. However, our proposed approach was not tested on the YOLO-LITE model. The YOLOv3 model was trained on a YOLO-labeled dataset with DarkNet-53. YOLOv3 has more tightness when it comes to bounding boxes and generally is more accurate than YOLO-LITE [53].

Moreover, the aim of our study was not to apply real-time hand gestures on limited computing power devices, since we collected hand images of different size and used different angles for diversity. Hence, we focused only on performance criteria rather than YOLO-LITE [54] criteria to recognize hand gestures of speed. Complex applications such as communication with deaf people, gesture-based signaling systems [55], sign language recognition [4], special signal languages used in sports [56], human action recognition [7], posture detection [57], physical exercise monitoring [10], and controlling smart homes for assisted living [58], are where GPUs could perform better through YOLOv3.

In future work we can apply mixed YOLOv3–LITE [59] on our datasets and new datasets of 1–10 numbers for all kind of applications for GPU and non-GPU based computers to achieve real-time object detection precisely and quickly. Furthermore, we can have enhanced our images through oversampling and real-time augmentation [60] as well. The major contribution is that there is no such dataset available in YOLO-labeled format and the research on, specifically, YOLOv3 and onwards requires the YOLO-labeled dataset so by our contribution that dataset will be readily available for future research and improvement as the domain of hand gesture recognition is very much wide there is a need of dataset that should be readily available in the YOLO format also.

6. Conclusions

In this paper, we have proposed a lightweight model based on the YOLOv3 and DarkNet-53 deep learning models for hand gesture recognition. The developed hand gesture recognition system detects both real-time objects and gestures from video frames with an accuracy of 97.68%. Despite the accuracy obtained there is still room for improvement in the following model, as right now the model proposed detects static gestures. However, the model can be improved for detecting multiple gestures and can be improved by detecting more than one gesture at a time. The proposed method can be used for improving assisted living systems, which are used for human–computer interaction both by healthy and impaired people. Additionally, we compared the performance and execution of the YOLOv3 model with different methods and our proposed method achieved better results by extracting features from the hand and recognized hand gestures with the accuracy, precision, recall, and F-1 score of 97.68, 94.88, 98.66, and 96.70%, respectively. For future work, we will be focusing on hybrid methods with smart mobile applications or robotics with different scenarios. Furthermore, we will design a more advanced convolution neural network with data fusion, inspired by recent works [61–63], to enhance the precision of hand gesture recognition.

Author Contributions: Conceptualization, A.M., M.J.A., A.Y., M.A.M., R.D. and K.H.A.; methodology, A.M., M.J.A., A.Y., M.A.M., R.D. and K.H.A.; software, A.M., M.J.A. and A.Y.; validation, M.J.A.; formal analysis, M.A.M.; writing—original draft preparation, A.M., M.J.A. and A.Y.; writing—review and editing, A.M, M.J.A., A.Y., M.A.M., R.D., R.M. and K.H.A.; supervision, M.A.M.; funding acquisition, R.D. and R.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable for this study.

Informed Consent Statement: Not applicable for this study.

Data Availability Statement: The dataset used is collected and labeled by the authors and is available at Google Drive at: https://drive.google.com/drive/folders/1X4q59iAUcPDY9lsJaq2HMb0GLU6cKp-q?usp=sharing (accessed on 1 February 2021).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Fang, Y.; Wang, K.; Cheng, J.; Lu, H. A Real-Time Hand Gesture Recognition Method. In Proceedings of the Multimedia and Expo, IEEE International Conference On Multimedia and Expo, Beijing, China, 2–5 July 2007. [CrossRef]
2. Oudah, M.; Al-Naji, A.; Chahl, J. Hand Gesture Recognition Based on Computer Vision: A Review of Techniques. *J. Imaging* **2020**, *6*, 73. [CrossRef]
3. Al-Hammadi, M.; Muhammad, G.; Abdul, W.; Alsulaiman, M.; Bencherif, M.A.; Alrayes, T.S.; Mekhtiche, M.A. Deep learning-based approach for sign language gesture recognition with efficient hand gesture representation. *IEEE Access* **2020**, *8*, 192527–192542. [CrossRef]
4. Vaitkevičius, A.; Taroza, M.; Blažauskas, T.; Damaševičius, R.; Maskeliūnas, R.; Woźniak, M. Recognition of american sign language gestures in a virtual reality using leap motion. *Appl. Sci.* **2019**, *9*, 445. [CrossRef]
5. Rezende, T.M.; Almeida, S.G.M.; Guimarães, F.G. Development and validation of a brazilian sign language database for human gesture recognition. *Neural Comput. Appl.* **2021**. [CrossRef]
6. Žemgulys, J.; Raudonis, V.; Maskeliūnas, R.; Damaševičius, R. Recognition of basketball referee signals from real-time videos. *J. Ambient Intell. Humaniz. Comput.* **2020**, *11*, 979–991. [CrossRef]
7. Afza, F.; Khan, M.A.; Sharif, M.; Kadry, S.; Manogaran, G.; Saba, T.; Ashraf, I.; Damaševičius, R. A framework of human action recognition using length control features fusion and weighted entropy-variances based feature selection. *Image Vision Comput.* **2021**, *106*, 104090. [CrossRef]
8. Nikolaidis, A.; Pitas, I. Facial feature extraction and pose determination. *Pattern Recognit.* **2000**, *33*, 1783–1791. [CrossRef]
9. Kulikajevas, A.; Maskeliunas, R.; Damaševičius, R. Detection of sitting posture using hierarchical image composition and deep learning. *PeerJ Comput. Sci.* **2021**, *7*, e442. [CrossRef] [PubMed]
10. Ryselis, K.; Petkus, T.; Blažauskas, T.; Maskeliūnas, R.; Damaševičius, R. Multiple kinect based system to monitor and analyze key performance indicators of physical training. *Hum. Centric Comput. Inf. Sci.* **2020**, *10*, 51. [CrossRef]
11. Huu, P.N.; Minh, Q.T.; The, H.L. An ANN-based gesture recognition algorithm for smart-home applications. *KSII Trans. Internet Inf. Syst.* **2020**, *14*, 1967–1983. [CrossRef]
12. Abraham, L.; Urru, A.; Normani, N.; Wilk, M.P.; Walsh, M.; O'Flynn, B. Hand Tracking and Gesture Recognition Using Lensless Smart Sensors. *Sensors* **2018**, *18*, 2834. [CrossRef]
13. Ahmed, S.; Cho, S.H. Hand Gesture Recognition Using an IR-UWB Radar with an Inception Module-Based Classifier. *Sensors* **2020**, *20*, 564. [CrossRef]
14. Lee, D.-H.; Kwang-Seok Hong, K.-S. Game interface using hand gesture recognition. In Proceedings of the 5th International Conference on Computer Sciences and Convergence Information Technology, Seoul, Korea, 30 November–2 December 2010. [CrossRef]
15. Alkemade, R.; Verbeek, F.J.; Lukosch, S.G. On the efficiency of a VR hand gesture-based interface for 3D object manipulations in conceptual design. *Int. J. Hum. Comput. Interact.* **2017**, *33*, 882–901. [CrossRef]
16. Lee, Y.S.; Sohn, B. Immersive gesture interfaces for navigation of 3D maps in HMD-based mobile virtual environments. *Mob. Inf. Syst.* **2018**, *2018*, 2585797. [CrossRef]
17. Del Rio Guerra, M.S.; Martin-Gutierrez, J.; Acevedo, R.; Salinas, S. Hand gestures in virtual and augmented 3D environments for down syndrome users. *Appl. Sci.* **2019**, *9*, 2641. [CrossRef]
18. Moschetti, A.; Fiorini, L.; Esposito, D.; Dario, P.; Cavallo, F. Toward an unsupervised approach for daily gesture recognition in assisted living applications. *IEEE Sens. J.* **2017**, *17*, 8395–8403. [CrossRef]
19. Mezari, A.; Maglogiannis, I. An easily customized gesture recognizer for assisted living using commodity mobile devices. *J. Healthc. Eng.* **2018**, *2018*, 3180652. [CrossRef] [PubMed]
20. Negin, F.; Rodriguez, P.; Koperski, M.; Kerboua, A.; Gonzàlez, J.; Bourgeois, J.; Bremond, F. PRAXIS: Towards automatic cognitive assessment using gesture recognition. *Expert Syst. Appl.* **2018**, *106*, 21–35. [CrossRef]
21. Kaczmarek, W.; Panasiuk, J.; Borys, S.; Banach, P. Industrial robot control by means of gestures and voice commands in off-line and on-line mode. *Sensors* **2020**, *20*, 6358. [CrossRef]
22. Neto, P.; Simão, M.; Mendes, N.; Safeea, M. Gesture-based human-robot interaction for human assistance in manufacturing. *Int. J. Adv. Manuf. Technol.* **2019**, *101*, 119–135. [CrossRef]
23. Young, G.; Milne, H.; Griffiths, D.; Padfield, E.; Blenkinsopp, R.; Georgiou, O. Designing mid-air haptic gesture controlled user interfaces for cars. In Proceedings of the ACM on Human-Computer Interaction, 4(EICS), Article No. 81, Honolulu, HI, USA, 25–30 April 2020. [CrossRef]
24. Yu, H.; Fan, X.; Zhao, L.; Guo, X. A novel hand gesture recognition method based on 2-channel sEMG. *Technol. Health Care* **2018**, *26*, 205–214. [CrossRef] [PubMed]
25. Zhao, L.; Li, S. Object detection algorithm based on improved YOLOv3. *Electronics* **2020**, *9*, 537. [CrossRef]
26. Kulikajevas, A.; Maskeliūnas, R.; Damaševičius, R.; Ho, E.S.L. 3D object reconstruction from imperfect depth data using extended yolov3 network. *Sensors* **2020**, *20*, 2025. [CrossRef]
27. Ni, Z.; Chen, J.; Sang, N.; Gao, C.; Liu, L. Light YOLO for High-Speed Gesture Recognition. In Proceedings of the 25th IEEE International Conference on Image Processing (ICIP), Athens, Greece, 7–10 October 2018. [CrossRef]

28. Chen, L.; Fu, J.; Wu, Y.; Li, H.; Zheng, B. Hand Gesture Recognition Using Compact CNN via Surface Electromyography Signals. *Sensors* **2020**, *20*, 672. [CrossRef] [PubMed]
29. Colli-Alfaro, J.G.; Ibrahim, A.; Trejos, A.L. Design of User-Independent Hand Gesture Recognition Using Multilayer Perceptron Networks and Sensor Fusion Techniques. In Proceedings of the IEEE 16th International Conference on Rehabilitation Robotics (ICORR), Toronto, ON, Canada, 24–28 June 2019; pp. 1103–1108. [CrossRef]
30. Elmezain, M.; Al-Hamadi, A.; Appenrodt, J.; Michaelis, B. A hidden markov model-based isolated and meaningful hand gesture recognition. *Int. J. Electr. Comput. Syst. Eng.* **2009**, *3*, 156–163.
31. Nyirarugira, C.; Choi, H.-R.; Kim, J.; Hayes, M.; Kim, T. Modified levenshtein distance for real-time gesture recognition. In Proceedings of the 6th International Congress on Image and Signal Processing (CISP), Hangzhou, China, 16–18 December 2013. [CrossRef]
32. Albawi, S.; Bayat, O.; Al-Azawi, S.; Ucan, O.N. Social Touch Gesture Recognition Using Convolutional Neural Network. *Comput. Intell. Neurosci.* **2018**, 1–10. [CrossRef]
33. Ju, M.; Luo, H.; Wang, Z.; Hui, B.; Chang, Z. The Application of Improved YOLO V3 in Multi-Scale Target Detection. *Appl. Sci.* **2019**, *9*, 3775. [CrossRef]
34. Saqib, S.; Ditta, A.; Khan, M.A.; Kazmi, S.A.R.; Alquhayz, H. Intelligent dynamic gesture recognition using CNN empowered by edit distance. *Comput. Mater. Contin.* **2020**, *66*, 2061–2076. [CrossRef]
35. Al-Hammadi, M.; Muhammad, G.; Abdul, W.; Alsulaiman, M.; Bencherif, M.A.; Mekhtiche, M.A. Hand gesture recognition for sign language using 3DCNN. *IEEE Access* **2020**, *8*, 79491–79509. [CrossRef]
36. Do, N.; Kim, S.; Yang, H.; Lee, G. Robust hand shape features for dynamic hand gesture recognition using multi-level feature LSTM. *Appl. Sci.* **2020**, *10*, 6293. [CrossRef]
37. Elboushaki, A.; Hannane, R.; Afdel, K.; Koutti, L. MultiD-CNN: A multi-dimensional feature learning approach based on deep convolutional networks for gesture recognition in RGB-D image sequences. *Expert Syst. Appl.* **2020**, *139*. [CrossRef]
38. Peng, Y.; Tao, H.; Li, W.; Yuan, H.; Li, T. Dynamic gesture recognition based on feature fusion network and variant ConvLSTM. *IET Image Process.* **2020**, *14*, 2480–2486. [CrossRef]
39. Tan, Y.S.; Lim, K.M.; Lee, C.P. Hand gesture recognition via enhanced densely connected convolutional neural network. *Expert Syst. Appl.* **2021**, *175*. [CrossRef]
40. Tran, D.; Ho, N.; Yang, H.; Baek, E.; Kim, S.; Lee, G. Real-time hand gesture spotting and recognition using RGB-D camera and 3D convolutional neural network. *Appl. Sci.* **2020**, *10*, 722. [CrossRef]
41. Rahim, M.A.; Islam, M.R.; Shin, J. Non-Touch Sign Word Recognition Based on Dynamic Hand Gesture Using Hybrid Segmentation and CNN Feature Fusion. *Appl. Sci.* **2019**, *9*, 3790. [CrossRef]
42. Mambou, S.; Krejcar, O.; Maresova, P.; Selamat, A.; Kuca, K. Novel Hand Gesture Alert System. *Appl. Sci.* **2019**, *9*, 3419. [CrossRef]
43. Ashiquzzaman, A.; Lee, H.; Kim, K.; Kim, H.-Y.; Park, J.; Kim, J. Compact Spatial Pyramid Pooling Deep Convolutional Neural Network Based Hand Gestures Decoder. *Appl. Sci.* **2020**, *10*, 7898. [CrossRef]
44. Benitez-Garcia, G.; Prudente-Tixteco, L.; Castro-Madrid, L.C.; Toscano-Medina, R.; Olivares-Mercado, J.; Sanchez-Perez, G.; Villalba, L.J.G. Improving Real-Time Hand Gesture Recognition with Semantic Segmentation. *Sensors* **2021**, *21*, 356. [CrossRef] [PubMed]
45. Bradski, G. The OpenCV Library. *Dr Dobb's J. Softw. Tools* **2000**, *25*, 120–125.
46. Everingham, M.; Van Gool, L.; Williams, C.K.; Winn, J.; Zisserman, A. The pascal visual object classes (voc) challenge. *Int. J. Comput. Vis.* **2010**, *88*, 303–338. [CrossRef]
47. Redmon, J.; Farhadi, A. YOLOv3: An Incremental Improvement. *arXiv* **2018**, arXiv:1804.02767.
48. Qassim, H.; Verma, A.; Feinzimer, D. Compressed residual-VGG16 CNN model for big data places image recognition. In Proceedings of the 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 8–10 January 2018; pp. 169–175. [CrossRef]
49. Fu, C.; Liu, W.; Ranga, A.; Tyagi, A.; Berg, A.C. DSSD: Deconvolutional Single Shot Detector. *arXiv* **2017**, arXiv:1701.06659.
50. Fong, S.; Liang, J.; Fister, I.; Fister, I.; Mohammed, S. Gesture Recognition from Data Streams of Human Motion Sensor Using Accelerated PSO Swarm Search Feature Selection Algorithm. *J. Sens.* **2015**, *2015*, 205707. [CrossRef]
51. Yan, S.; Xia, Y.; Smith, J.S.; Lu, W.; Zhang, B. Multiscale Convolutional Neural Networks for Hand Detection. *Appl. Comput. Intell. Soft Comput.* **2017**, *2017*, 9830641. [CrossRef]
52. Ren, Z.; Yuan, J.; Meng, J.; Zhang, Z. Robust Part-Based Hand Gesture Recognition Using Kinect Sensor. *IEEE Trans. Multimed.* **2013**, *15*, 1110–1120. [CrossRef]
53. Pedoeem, J.; Huang, R. YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers. In Proceedings of the IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 10–13 December 2018; pp. 2503–2510.
54. Sismananda, P.; Abdurohman, M.; Putrada, A.G. Performance Comparison of Yolo-Lite and YoloV3 Using Raspberry Pi and MotionEyeOS. In Proceedings of the 8th International Conference on Information and Communication Technology (ICoICT), Yogyakarta, Indonesia, 4–5 August 2020; pp. 1–7.
55. Połap, D. Human-machine interaction in intelligent technologies using the augmented reality. *Inf. Technol. Control* **2018**, *47*, 691–703. [CrossRef]
56. Žemgulys, J.; Raudonis, V.; Maskeliunas, R.; Damaševičius, R. Recognition of basketball referee signals from videos using histogram of oriented gradients (HOG) and support vector machine (SVM). *Procedia Comput. Sci.* **2018**, *130*, 953–960. [CrossRef]

57. Wozniak, M.; Wieczorek, M.; Silka, J.; Polap, D. Body pose prediction based on motion sensor data and recurrent neural network. *IEEE Trans. Ind. Inform.* **2021**, *17*, 2101–2111. [CrossRef]
58. Maskeliunas, R.; Damaševicius, R.; Segal, S. A review of internet of things technologies for ambient assisted living environments. *Future Internet* **2019**, *11*, 259. [CrossRef]
59. Zhao, H.; Zhou, Y.; Zhang, L.; Peng, Y.; Hu, X.; Peng, H.; Cai, X. Mixed YOLOv3-LITE: A Lightweight Real-Time Object Detection Method. *Sensors* **2020**, *20*, 1861. [CrossRef]
60. Awan, M.J.; Rahim, M.S.M.; Salim, N.; Mohammed, M.A.; Garcia-Zapirain, B.; Abdulkareem, K.H. Efficient Detection of Knee Anterior Cruciate Ligament from Magnetic Resonance Imaging Using Deep Learning Approach. *Diagnostics* **2021**, *11*, 105. [CrossRef] [PubMed]
61. Mastoi, Q.; Memon, M.S.; Lakhan, A.; Mohammed, M.A.; Qabulio, M.; Al-Turjman, F.; Abdulkareem, K.H. Machine learning-data mining integrated approach for premature ventricular contraction prediction. *Neural Comput. Appl.* **2021**. [CrossRef]
62. Mohammed, M.A.; Abdulkareem, K.H.; Mostafa, S.A.; Ghani, M.K.A.; Maashi, M.S.; Garcia-Zapirain, B.; Oleagordia, I.; Alhakami, H.; Al-Dhief, F.T. Voice pathology detection and classification using convolutional neural network model. *Appl. Sci.* **2020**, *10*, 3723. [CrossRef]
63. Kashinath, S.A.; Mostafa, S.A.; Mustapha, A.; Mahdin, H.; Lim, D.; Mahmoud, M.A.; Mohammed, M.A.; Al-Rimy, B.A.S.; Fudzee, M.F.M.; Yang, T.J. Review of Data Fusion Methods for Real-Time and Multi-Sensor Traffic Flow Analysis. *IEEE Access* **2021**, *9*, 51258–51276. [CrossRef]

Appendix B: Sample Code

Main project code with gui

```
import mysql.connector as mcon
from tkinter import *
from tkinter import ttk
import tkinter.messagebox
from PIL import Image, ImageTk

con = mcon.connect(host="localhost", user="root",
password="1234")
c = con.cursor()
c.execute("use ges")

class Table:
    def __init__(self, root, total_rows, total_columns,
lst):
        v = Scrollbar(root, orient=VERTICAL)
        v.pack(side=RIGHT, fill=Y)

        view2 = Frame(root, bg='#2B65EC', width=1920,
height=500)
        view2.pack(side=LEFT, fill=Y)

        # code for creating table
        for i in range(total_rows):
            for j in range(total_columns):
                self.e = Label(view2, width=20,
fg='black', text=str(lst[i][j]),
                               font=('Arial', 16, 'bold'),
relief=SOLID, borderwidth=2)
                self.e.grid(row=i, column=j)

def view_all_details():
    global view, main, image
    main.destroy()
    view = Frame(root, bg='#2B65EC', width=1920,
height=1080)
    view.pack()
    image = Image.open("gtov.png")
    image = image.resize((1920, 1080), Image.ANTIALIAS)
    image = ImageTk.PhotoImage(image)
    background_label = Label(view, image=image)
```

```
    background_label.image = image
    background_label.pack(fill="both", expand=True)
    label1 = Label(view, text='All requests', font=('semi
bold', 40, 'normal'), bg='#2B65EC')
    label1.place(x=700, y=128)
    view1 = Frame(root, bg='#2B65EC', width=1920,
height=500)
    view1.place(x=100, y=250)
    back = Button(view, text='BACK', width=5, font=('semi
bold', 14, 'normal'), command=main_area)
    back["highlightthickness"] = 5
    back.place(x=25, y=25)
    c.execute("select * from gesture")
    lst = c.fetchall()
    total_rows = len(lst)
    total_columns = len(lst[0])
    Table(view1, total_rows, total_columns, lst)

def search():
    global view, main, image
    main.destroy()
    view = Frame(root, bg='#2B65EC', width=1920,
height=1080)
    view.pack()
    image = PhotoImage(file="gtov.png")
    background_label = Label(view, image=image)
    background_label.pack(fill="both", expand=True)
    label1 = Label(view, text='SEARCH', font=('semi bold',
40, 'normal'))
    label1.place(x=700, y=128)
    e1 = Entry(view, width=15, font=('semi bold', 18,
'normal'))
    e1.insert(0, "from: yyyy-mm-dd")
    e1.place(x=700, y=228)
    e2 = Entry(view, width=15, font=('semi bold', 18,
'normal'))
    e2.insert(0, "to: yyyy-mm-dd")
    e2.place(x=700, y=328)
    view1 = Frame(root, bg='#2B65EC', width=600,
height=500)
    view1.place(x=30, y=180)
```

```
    back = Button(view, text='BACK', width=20, font=('semi
bold', 20, 'normal'), command=main_area)
    back.place(x=100, y=100)

    def find():
     c.execute("SELECT * FROM gesture WHERE tim > %s AND
tim < %s", (e1.get(), e2.get()))
     lst = c.fetchall()
     total_rows = len(lst)
     total_columns = len(lst[0])

     Table(view1, total_rows, total_columns, lst)



    b1 = Button(view, text='SEARCH', width=20, font=('semi
bold', 20, 'normal'), command=find)
    b1.place(x=700, y=400)


def run():
    import project1 as p
    p.main()

def Hp():
    global view, main, image
    main.destroy()
    view = Frame(root, bg='#2B65EC', width=1920,
height=1080)
    view.pack()
    image = Image.open("help.png")
    image = image.resize((1220, 799), Image.ANTIALIAS)
    image = ImageTk.PhotoImage(image)
    background_label = Label(view, image=image)
    background_label.image = image
    background_label.pack(fill="both", expand=True)
    back = Button(view, text='BACK', width=5, font=('semi
bold', 14, 'normal'), command=main_area)
    back["highlightthickness"] = 5
    back.place(x=25, y=25)
def main_area():
    global main, view, image
```

```
    view.destroy()
    main = Frame(root, bg='#2B65EC', width=1920,
height=1080)
    main.pack()
    image = Image.open("gtov.png")
    image = image.resize((1920, 1080), Image.ANTIALIAS)
    image = ImageTk.PhotoImage(image)
    background_label = Label(main, image=image)
    background_label.image = image
    background_label.pack(fill="both", expand=True)
    b1 = Button(main, text='VIEW ALL DETAILS', width=20,
font=('semi bold', 20, 'normal'),
command=view_all_details)
    b1.place(x=700, y=128)
    b2 = Button(main, text='SEARCH', width=20, font=('semi
bold', 20, 'normal'), command=search)
    b2.place(x=700, y=228)
    b3 = Button(main, text='Execute', width=20,
font=('semi bold', 20, 'normal'), command=run)
    b3.place(x=700, y=328)
    b4 = Button(main, text='Help', width=20, font=('semi
bold', 20, 'normal'), command=Hp)
    b4.place(x=700, y=428)
    b5 = Button(main, text='logout', width=20, font=('semi
bold', 20, 'normal'), command=log)
    b5.place(x=700, y=528)

def logincheck():
    global pid, user_name
    user_name = l1.get()
    password = l2.get()
    c.execute("select * from login where
name='"+user_name+"'and password='"+password+"'")
    d = c.fetchall()
    pid = d[0][0]
    if len(d) != 0:
        print('login successful')
        login.destroy()
        main_area()
    else:
        tkinter.messagebox.showinfo("ERROR", "Incorrect
credentials")
```

```
        print('login unsuccessful')

root = Tk()
root.geometry("1920x1080")
root.title("gesture to voice activation")
view = Frame(root, bg='#2B65EC', width=1920, height=1080)
main = Frame(root, bg='#2B65EC', width=1920, height=1080)

def log():
    global view, login, image,main,l1,l2
    main.destroy()
    view.destroy()
    login = Frame(root, bg='#2B65EC', width=1920,
height=1080)
    login.pack()
    image = Image.open("gtov.png")
    image = image.resize((1920, 1080), Image.ANTIALIAS)
    image = ImageTk.PhotoImage(image)
    background_label = Label(login, image=image)
    background_label.image = image
    background_label.pack(fill="both", expand=True)
    view = Frame(root, bg='#2B65EC', width=1920,
height=1080)
    l1 = Entry(login, width=15, font=('semi bold', 40,
'normal'))
    l1.place(x=750, y=228)
    l2 = Entry(login, width=15, show="*", font=('semi
bold', 40, 'normal'))
    l2.place(x=750, y=328)
    label1 = Label(login, text='LOGIN', font=('semi bold',
40, 'normal'))
    label1.place(x=900, y=128)
    label1 = Label(login, text='USER NAME:', font=('semi
bold', 40, 'normal'), bg='#2B65EC')
    label1.place(x=400, y=228)
    label1 = Label(login, text='PASSWORD:', font=('semi
bold', 40, 'normal'), bg='#2B65EC')
    label1.place(x=400, y=328)
    loginbutton = Button(login, width=15, height=1,
font=('semi bold', 35, 'normal'), bg='#C72542',
text='LOGIN', command=logincheck)
    loginbutton.place(x=760, y=450)
```

```
    def show_pswd():
       if l2.cget("show") == "*":
          l2.config(show='')
       else:
          l2.config(show='*')

    c_button = Checkbutton(root, text="show password",
font=('semi bold', 24, 'normal'), command=show_pswd)
    c_button.place(x=750, y=395)

log()
root.mainloop()
```

Gesture detection

```
import csv
import copy
import argparse
import itertools
from collections import Counter
from collections import deque
import mysql.connector as mysql
import cv2 as cv
import numpy as np
import mediapipe as mp
import pyttsx3
from twilio.rest import Client
import time
import pygame

from utils import CvFpsCalc
from model import KeyPointClassifier
from model import PointHistoryClassifier
from datetime import datetime
now = datetime.now()
formatted_date = now.strftime('%Y-%m-%d %H:%M:%S')

engine=pyttsx3.init()  #object creation
voices=engine.getProperty('voices')   #getting details of
current voice
engine.setProperty('rate', 206)     #voice speed
```

```
engine.setProperty('volume', 100)   # volume

#engine.setProperty('voice',voices[0].id)  #male voice

engine.setProperty('voice',voices[0].id)

account_sid = 'ACb0a96e090a0eb0b5a62337876b1f685d'
auth_token = 'a1536e26a632846139ecbf2a3a4bde03'
client = Client(account_sid, auth_token)


def get_args():
    parser = argparse.ArgumentParser()

    parser.add_argument("--device", type=int, default=0)
    parser.add_argument("--width", help='cap width',
type=int, default=960)
    parser.add_argument("--height", help='cap height',
type=int, default=540)

    parser.add_argument('--use_static_image_mode',
action='store_true')
    parser.add_argument("--min_detection_confidence",
                        help='min_detection_confidence',
                        type=float,
                        default=0.7)
    parser.add_argument("--min_tracking_confidence",
                        help='min_tracking_confidence',
                        type=int,
                        default=0.5)

    args = parser.parse_args()

    return args


def main():
    # Argument parsing
    args = get_args()

    cap_device = args.device
    cap_width = args.width
    cap_height = args.height
```

```
    use_static_image_mode = args.use_static_image_mode
    min_detection_confidence =
args.min_detection_confidence
    min_tracking_confidence = args.min_tracking_confidence

    use_brect = True

    # Camera preparation
    cap = cv.VideoCapture(cap_device)
    cap.set(cv.CAP_PROP_FRAME_WIDTH, cap_width)
    cap.set(cv.CAP_PROP_FRAME_HEIGHT, cap_height)

    # Model load
    mp_hands = mp.solutions.hands
    hands = mp_hands.Hands(
        static_image_mode=use_static_image_mode,
        max_num_hands=2,
        min_detection_confidence=min_detection_confidence,
        min_tracking_confidence=min_tracking_confidence,
    )

    keypoint_classifier = KeyPointClassifier()

    point_history_classifier = PointHistoryClassifier()

    # Read labels
    with
open('model/keypoint_classifier/keypoint_classifier_label.
csv',
              encoding='utf-8-sig') as f:
        keypoint_classifier_labels = csv.reader(f)
        keypoint_classifier_labels = [
            row[0] for row in keypoint_classifier_labels
        ]
    with open(
            'model/point_history_classifier/point_history_
classifier_label.csv',
            encoding='utf-8-sig') as f:
        point_history_classifier_labels = csv.reader(f)
        point_history_classifier_labels = [
```

```
            row[0] for row in
point_history_classifier_labels
        ]

    # FPS Measurement
    cvFpsCalc = CvFpsCalc(buffer_len=10)

    # Coordinate history
    history_length = 16
    point_history = deque(maxlen=history_length)

    # Finger gesture history
    finger_gesture_history = deque(maxlen=history_length)


    mode = 0

    while True:
        fps = cvFpsCalc.get()

        # Process Key (ESC: end)
        key = cv.waitKey(10)
        if key == 27:  # ESC
            break
        number, mode = select_mode(key, mode)

        # Camera capture
        ret, image = cap.read()
        if not ret:
            break
        image = cv.flip(image, 1)  # Mirror display
        debug_image = copy.deepcopy(image)

        # Detection implementation
        image = cv.cvtColor(image, cv.COLOR_BGR2RGB)

        image.flags.writeable = False
        results = hands.process(image)
        image.flags.writeable = True

        if results.multi_hand_landmarks is not None:
```

```
            for hand_landmarks, handedness in
zip(results.multi_hand_landmarks,
                                                results.
multi_handedness):
                # Bounding box calculation
                brect = calc_bounding_rect(debug_image,
hand_landmarks)
                # Landmark calculation
                landmark_list =
calc_landmark_list(debug_image, hand_landmarks)

                # Conversion to relative coordinates /
normalized coordinates
                pre_processed_landmark_list =
pre_process_landmark(
                    landmark_list)
                pre_processed_point_history_list =
pre_process_point_history(
                    debug_image, point_history)
                # Write to the dataset file
                logging_csv(number, mode,
pre_processed_landmark_list,
                            pre_processed_point_history_li
st)

                # Hand sign classification
                hand_sign_id =
keypoint_classifier(pre_processed_landmark_list)
                if hand_sign_id == 2:  # Point gesture
                    point_history.append(landmark_list[8])
                else:
                    point_history.append([0, 0])

                # Finger gesture classification
                finger_gesture_id = 0
                point_history_len =
len(pre_processed_point_history_list)
                if point_history_len == (history_length *
2):
                    finger_gesture_id =
point_history_classifier(
                        pre_processed_point_history_list)
```

```
                # Calculates the gesture IDs in the latest
detection
                finger_gesture_history.append(finger_gestu
re_id)
                most_common_fg_id = Counter(
                    finger_gesture_history).most_common()

                # Drawing part
                debug_image =
draw_bounding_rect(use_brect, debug_image, brect)
                debug_image = draw_landmarks(debug_image,
landmark_list)
                debug_image = draw_info_text(
                    debug_image,
                    brect,
                    handedness,
                    keypoint_classifier_labels[hand_sign_i
d],
                    point_history_classifier_labels[most_c
ommon_fg_id[0][0]],
                )
        else:
            point_history.append([0, 0])

        debug_image = draw_point_history(debug_image,
point_history)
        debug_image = draw_info(debug_image, fps, mode,
number)

        # Screen reflection
        cv.imshow('Hand Gesture Recognition', debug_image)

       # time.sleep(20)
    cap.release()
    cv.destroyAllWindows()


def select_mode(key, mode):
    number = -1
    if 48 <= key <= 57:  # 0 ~ 9
        number = key - 48
```

```
    if key == 110:  # n
        mode = 0
    if key == 107:  # k
        mode = 1
    if key == 104:  # h
        mode = 2
    return number, mode


def calc_bounding_rect(image, landmarks):
    image_width, image_height = image.shape[1],
image.shape[0]

    landmark_array = np.empty((0, 2), int)

    for _, landmark in enumerate(landmarks.landmark):
        landmark_x = min(int(landmark.x * image_width),
image_width - 1)
        landmark_y = min(int(landmark.y * image_height),
image_height - 1)

        landmark_point = [np.array((landmark_x,
landmark_y))]

        landmark_array = np.append(landmark_array,
landmark_point, axis=0)

    x, y, w, h = cv.boundingRect(landmark_array)

    return [x, y, x + w, y + h]


def calc_landmark_list(image, landmarks):
    image_width, image_height = image.shape[1],
image.shape[0]

    landmark_point = []

    # Keypoint
    for _, landmark in enumerate(landmarks.landmark):
        landmark_x = min(int(landmark.x * image_width),
image_width - 1)
```

```
        landmark_y = min(int(landmark.y * image_height),
image_height - 1)
        # landmark_z = landmark.z

        landmark_point.append([landmark_x, landmark_y])

    return landmark_point


def pre_process_landmark(landmark_list):
    temp_landmark_list = copy.deepcopy(landmark_list)

    # Convert to relative coordinates
    base_x, base_y = 0, 0
    for index, landmark_point in
enumerate(temp_landmark_list):
        if index == 0:
            base_x, base_y = landmark_point[0],
landmark_point[1]

        temp_landmark_list[index][0] =
temp_landmark_list[index][0] - base_x
        temp_landmark_list[index][1] =
temp_landmark_list[index][1] - base_y

    # Convert to a one-dimensional list
    temp_landmark_list = list(
        itertools.chain.from_iterable(temp_landmark_list))

    # Normalization
    max_value = max(list(map(abs, temp_landmark_list)))

    def normalize_(n):
        return n / max_value

    temp_landmark_list = list(map(normalize_,
temp_landmark_list))

    return temp_landmark_list


def pre_process_point_history(image, point_history):
```

```
    image_width, image_height = image.shape[1],
image.shape[0]

    temp_point_history = copy.deepcopy(point_history)

    # Convert to relative coordinates
    base_x, base_y = 0, 0
    for index, point in enumerate(temp_point_history):
        if index == 0:
            base_x, base_y = point[0], point[1]

        temp_point_history[index][0] =
(temp_point_history[index][0] -
                                        base_x) /
image_width
        temp_point_history[index][1] =
(temp_point_history[index][1] -
                                        base_y) /
image_height

    # Convert to a one-dimensional list
    temp_point_history = list(
        itertools.chain.from_iterable(temp_point_history))

    return temp_point_history


def logging_csv(number, mode, landmark_list,
point_history_list):
    if mode == 0:
        pass
    if mode == 1 and (0 <= number <= 9):
        csv_path =
'model/keypoint_classifier/keypoint.csv'
        with open(csv_path, 'a', newline="") as f:
            writer = csv.writer(f)
            writer.writerow([number, *landmark_list])
    if mode == 2 and (0 <= number <= 9):
        csv_path =
'model/point_history_classifier/point_history.csv'
        with open(csv_path, 'a', newline="") as f:
            writer = csv.writer(f)
```

```
            writer.writerow([number, *point_history_list])
    return


def draw_landmarks(image, landmark_point):
    if len(landmark_point) > 0:
        # Thumb
        cv.line(image, tuple(landmark_point[2]),
tuple(landmark_point[3]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[2]),
tuple(landmark_point[3]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[3]),
tuple(landmark_point[4]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[3]),
tuple(landmark_point[4]),
                (255, 255, 255), 2)

        # Index finger
        cv.line(image, tuple(landmark_point[5]),
tuple(landmark_point[6]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[5]),
tuple(landmark_point[6]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[6]),
tuple(landmark_point[7]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[6]),
tuple(landmark_point[7]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[7]),
tuple(landmark_point[8]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[7]),
tuple(landmark_point[8]),
                (255, 255, 255), 2)

        # Middle finger
```

```
        cv.line(image, tuple(landmark_point[9]),
tuple(landmark_point[10]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[9]),
tuple(landmark_point[10]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[10]),
tuple(landmark_point[11]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[10]),
tuple(landmark_point[11]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[11]),
tuple(landmark_point[12]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[11]),
tuple(landmark_point[12]),
                (255, 255, 255), 2)

        # Ring finger
        cv.line(image, tuple(landmark_point[13]),
tuple(landmark_point[14]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[13]),
tuple(landmark_point[14]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[14]),
tuple(landmark_point[15]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[14]),
tuple(landmark_point[15]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[15]),
tuple(landmark_point[16]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[15]),
tuple(landmark_point[16]),
                (255, 255, 255), 2)

        # Little finger
        cv.line(image, tuple(landmark_point[17]),
tuple(landmark_point[18]),
```

```
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[17]),
tuple(landmark_point[18]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[18]),
tuple(landmark_point[19]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[18]),
tuple(landmark_point[19]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[19]),
tuple(landmark_point[20]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[19]),
tuple(landmark_point[20]),
                (255, 255, 255), 2)

        # Palm
        cv.line(image, tuple(landmark_point[0]),
tuple(landmark_point[1]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[0]),
tuple(landmark_point[1]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[1]),
tuple(landmark_point[2]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[1]),
tuple(landmark_point[2]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[2]),
tuple(landmark_point[5]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[2]),
tuple(landmark_point[5]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[5]),
tuple(landmark_point[9]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[5]),
tuple(landmark_point[9]),
                (255, 255, 255), 2)
```

```
        cv.line(image, tuple(landmark_point[9]),
tuple(landmark_point[13]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[9]),
tuple(landmark_point[13]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[13]),
tuple(landmark_point[17]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[13]),
tuple(landmark_point[17]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[17]),
tuple(landmark_point[0]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[17]),
tuple(landmark_point[0]),
                (255, 255, 255), 2)

    # Key Points
    for index, landmark in enumerate(landmark_point):
        if index == 0:
            cv.circle(image, (landmark[0], landmark[1]),
5, (255, 255, 255),
                      -1)
            cv.circle(image, (landmark[0], landmark[1]),
5, (0, 0, 0), 1)
        if index == 1:
            cv.circle(image, (landmark[0], landmark[1]),
5, (255, 255, 255),
                      -1)
            cv.circle(image, (landmark[0], landmark[1]),
5, (0, 0, 0), 1)
        if index == 2:
            cv.circle(image, (landmark[0], landmark[1]),
5, (255, 255, 255),
                      -1)
            cv.circle(image, (landmark[0], landmark[1]),
5, (0, 0, 0), 1)
        if index == 3:
            cv.circle(image, (landmark[0], landmark[1]),
5, (255, 255, 255),
```

```
                    -1)
            cv.circle(image, (landmark[0], landmark[1]),
5, (0, 0, 0), 1)
        if index == 4:
            cv.circle(image, (landmark[0], landmark[1]),
8, (255, 255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]),
8, (0, 0, 0), 1)
        if index == 5:
            cv.circle(image, (landmark[0], landmark[1]),
5, (255, 255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]),
5, (0, 0, 0), 1)
        if index == 6:
            cv.circle(image, (landmark[0], landmark[1]),
5, (255, 255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]),
5, (0, 0, 0), 1)
        if index == 7:
            cv.circle(image, (landmark[0], landmark[1]),
5, (255, 255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]),
5, (0, 0, 0), 1)
        if index == 8:
            cv.circle(image, (landmark[0], landmark[1]),
8, (255, 255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]),
8, (0, 0, 0), 1)
        if index == 9:
            cv.circle(image, (landmark[0], landmark[1]),
5, (255, 255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]),
5, (0, 0, 0), 1)
        if index == 10:
            cv.circle(image, (landmark[0], landmark[1]),
5, (255, 255, 255),
```

```
                    -1)
            cv.circle(image, (landmark[0], landmark[1]),
5, (0, 0, 0), 1)
        if index == 11:
            cv.circle(image, (landmark[0], landmark[1]),
5, (255, 255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]),
5, (0, 0, 0), 1)
        if index == 12:
            cv.circle(image, (landmark[0], landmark[1]),
8, (255, 255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]),
8, (0, 0, 0), 1)
        if index == 13:
            cv.circle(image, (landmark[0], landmark[1]),
5, (255, 255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]),
5, (0, 0, 0), 1)
        if index == 14:
            cv.circle(image, (landmark[0], landmark[1]),
5, (255, 255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]),
5, (0, 0, 0), 1)
        if index == 15:
            cv.circle(image, (landmark[0], landmark[1]),
5, (255, 255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]),
5, (0, 0, 0), 1)
        if index == 16:
            cv.circle(image, (landmark[0], landmark[1]),
8, (255, 255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]),
8, (0, 0, 0), 1)
        if index == 17:
            cv.circle(image, (landmark[0], landmark[1]),
5, (255, 255, 255),
```

```
                    -1)
            cv.circle(image, (landmark[0], landmark[1]),
5, (0, 0, 0), 1)
        if index == 18:
            cv.circle(image, (landmark[0], landmark[1]),
5, (255, 255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]),
5, (0, 0, 0), 1)
        if index == 19:
            cv.circle(image, (landmark[0], landmark[1]),
5, (255, 255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]),
5, (0, 0, 0), 1)
        if index == 20:
            cv.circle(image, (landmark[0], landmark[1]),
8, (255, 255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]),
8, (0, 0, 0), 1)

    return image


def draw_bounding_rect(use_brect, image, brect):
    if use_brect:
        # Outer rectangle
        cv.rectangle(image, (brect[0], brect[1]),
(brect[2], brect[3]),
                     (0, 0, 0), 1)

    return image


def draw_info_text(image, brect, handedness,
hand_sign_text,
                   finger_gesture_text):
    cv.rectangle(image, (brect[0], brect[1]), (brect[2],
brect[1] - 22),
                 (0, 0, 0), -1)

    info_text = handedness.classification[0].label[0:]
```

```
    if hand_sign_text != "":
        info_text = info_text + ':' + hand_sign_text
    cv.putText(image, info_text, (brect[0] + 5, brect[1] -
4),
               cv.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255,
255), 1, cv.LINE_AA)

    if finger_gesture_text != "":
        cv.putText(image, "Finger Gesture:" +
hand_sign_text, (10, 60),
                   cv.FONT_HERSHEY_SIMPLEX, 1.0, (0, 0,
0), 4, cv.LINE_AA)
        cv.putText(image, "Finger Gesture:" +
hand_sign_text, (10, 60),
                   cv.FONT_HERSHEY_SIMPLEX, 1.0, (255,
255, 255), 2,
                   cv.LINE_AA)
    text = hand_sign_text
    if text == "Emergency":
            # Initialize the pygame mixer
            pygame.mixer.init()

            # Load the audio file (replace
'audio_file.mp3' with the path to your audio file)
            audio_file_path = 'alarm.mp3'
            pygame.mixer.music.load(audio_file_path)

            # Play the audio file
            pygame.mixer.music.play()

            # Add a delay to allow the audio to finish
playing (adjust the delay time as needed)
            pygame.time.wait(5000)  # This will wait for 5
seconds (5000 milliseconds)

            # Stop the audio playback
            pygame.mixer.music.stop()

            # Quit pygame mixer
            pygame.mixer.quit()

    engine.say(text)
```

```
    engine.runAndWait()
    #twilio recovery code
89oco0UH2YGq8avlhPPfpDCLzObvGltdyae2EkP4
    #create table users(username varchar(10),password
varchar(10));
    #create table gesture (command varchar(20),tim
DATETIME);
    """
     SET @time := '1000-01-01 00:00:00';
SET @interval := 60;
SELECT command,tim
FROM gesture
WHERE TIMESTAMPDIFF( SECOND, @time, tim ) >= @interval
    AND @time := tim;
   """
    message = client.messages.create(
    from_='whatsapp:+14155238886',
    to='whatsapp:+918714838557',
    body=text
)

    message = client.messages.create(
    body=text,
    from_='+14326662651',
    to='+918714838557'
)

    conn = mysql.connect(host="localhost", user="root",
password="1234", database="ges")
    mycursor = conn.cursor()
    mycursor.execute("INSERT INTO gesture VALUES('" +
hand_sign_text + "','" + formatted_date + "')")
    mycursor.execute("commit");
    conn.close();

    return image


def draw_point_history(image, point_history):
    for index, point in enumerate(point_history):
        if point[0] != 0 and point[1] != 0:
```

```
            cv.circle(image, (point[0], point[1]), 1 +
int(index / 2),
                      (152, 251, 152), 2)

    return image


def draw_info(image, fps, mode, number):
    cv.putText(image, "FPS:" + str(fps), (10, 30),
cv.FONT_HERSHEY_SIMPLEX,
               1.0, (0, 0, 0), 4, cv.LINE_AA)
    cv.putText(image, "FPS:" + str(fps), (10, 30),
cv.FONT_HERSHEY_SIMPLEX,
               1.0, (255, 255, 255), 2, cv.LINE_AA)

    mode_string = ['Logging Key Point', 'Logging Point
History']
    if 1 <= mode <= 2:
        cv.putText(image, "MODE:" + mode_string[mode - 1],
(10, 90),
                   cv.FONT_HERSHEY_SIMPLEX, 0.6, (255,
255, 255), 1,
                   cv.LINE_AA)
        if 0 <= number <= 9:
            cv.putText(image, "NUM:" + str(number), (10,
110),
                       cv.FONT_HERSHEY_SIMPLEX, 0.6, (255,
255, 255), 1,
                       cv.LINE_AA)
    return image


if __name__ == '__main__':
    main()
```

Appendix C: CO-PO And CO-PSO Mapping

COURSE OUTCOMES:

After completion of the course the student will be able to

SL. NO	DESCRIPTION	Blooms' Taxonomy Level
CO1	Identify technically and economically feasible problems (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO2	Identify and survey the relevant literature for getting exposed to related solutions and get familiarized with software development processes (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO3	Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions of minimal complexity by using modern tools & advanced programming techniques (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO4	Prepare technical report and deliver presentation (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO5	Apply engineering and management principles to achieve the goal of the project (Cognitive Knowledge Level: Apply)	Level 3: Apply

CO-PO AND CO-PSO MAPPING

	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO3
CO1	3	3	3	3		2	2	3	2	2	2	3	2	2	2
CO2	3	3	3	3	3	2		3	2	3	2	3	2	2	2
CO3	3	3	3	3	3	2	2	3	2	2	2	3			2
CO4	2	3	2	2	2			3	3	3	2	3	2	2	2
CO5	3	3	3	2	2	2	2	3	2		2	3	2	2	2

3/2/1: high/medium/low

JUSTIFICATIONS FOR CO-PO MAPPING

MAPPING	LOW/ MEDIUM/ HIGH	JUSTIFICATION
100003/CS622T.1-PO1	**HIGH**	Identify technically and economically feasible problems by applying the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS622T.1-PO2	**HIGH**	Identify technically and economically feasible problems by analysing complex engineering problems reaching substantiated conclusions using first principles of mathematics.
100003/CS622T.1-PO3	**HIGH**	Design solutions for complex engineering problems by identifying technically and economically feasible problems.
100003/CS622T.1-PO4	**HIGH**	Identify technically and economically feasible problems by analysis and interpretation of data.
100003/CS622T.1-PO6	**MEDIUM**	Responsibilities relevant to the professional engineering practice by identifying the problem.
100003/CS622T.1-PO7	**MEDIUM**	Identify technically and economically feasible problems by understanding the impact of the professional engineering solutions.
100003/CS622T.1-PO8	**HIGH**	Apply ethical principles and commit to professional ethics to identify technically and economically feasible problems.
100003/CS622T.1-PO9	**MEDIUM**	Identify technically and economically feasible problems by working as a team.
100003/CS622T.1-PO10	**MEDIUM**	Communicate effectively with the engineering community by identifying technically and economically feasible problems.
100003/CS622T.1-P011	**MEDIUM**	Demonstrate knowledge and understanding of engineering and management principles by selecting the technically and economically feasible problems.
100003/CS622T.1-PO12	**HIGH**	Identify technically and economically feasible problems for long term learning.
100003/CS622T.1-PSO1	**MEDIUM**	Ability to identify, analyze and design solutions to identify technically and economically feasible problems.
100003/CS622T.1-PSO2	**MEDIUM**	By designing algorithms and applying standard practices in software project development and Identifying technically and economically feasible problems.
100003/CS622T.1-PSO3	**MEDIUM**	Fundamentals of computer science in competitive research can be applied to Identify technically and economically feasible problems.
100003/CS622T.2-PO1	**HIGH**	Identify and survey the relevant by applying the knowledge of mathematics, science, engineering fundamentals.

100003/CS622T.2-PO2	**HIGH**	Identify, formulate, review research literature, and analyze complex engineering problems get familiarized with software development processes.
100003/CS622T.2-PO3	**HIGH**	Design solutions for complex engineering problems and design based on the relevant literature.
100003/CS622T.2-PO4	**HIGH**	Use research-based knowledge including design of experiments based on relevant literature.
100003/CS622T.2-PO5	**HIGH**	Identify and survey the relevant literature for getting exposed to related solutions and get familiarized with software development processes by using modern tools.
100003/CS622T.2-PO6	**MEDIUM**	Create, select, and apply appropriate techniques, resources, by identifying and surveying the relevant literature.
100003/CS622T.2-PO8	**HIGH**	Apply ethical principles and commit to professional ethics based on the relevant literature.
100003/CS622T.2-PO9	**MEDIUM**	Identify and survey the relevant literature as a team.
100003/CS622T.2-PO10	**HIGH**	Identify and survey the relevant literature for a good communication to the engineering fraternity.
100003/CS622T.2-PO11	**MEDIUM**	Identify and survey the relevant literature to demonstrate knowledge and understanding of engineering and management principles.
100003/CS622T.2-PO12	**HIGH**	Identify and survey the relevant literature for independent and lifelong learning.
100003/CS622T.2-PSO1	**MEDIUM**	Design solutions for complex engineering problems by Identifying and survey the relevant literature.
100003/CS622T.2-PSO2	**MEDIUM**	Identify and survey the relevant literature for acquiring programming efficiency by designing algorithms and applying standard practices.
100003/CS622T.2-PSO3	**MEDIUM**	Identify and survey the relevant literature to apply the fundamentals of computer science in competitive research.
100003/CS622T.3-PO1	**HIGH**	Perform requirement analysis, identify design methodologies by using modern tools & advanced programming techniques and by applying the knowledge of mathematics, science, engineering fundamentals.
100003/CS622T.3-PO2	**HIGH**	Identify, formulate, review research literature for requirement analysis, identify design methodologies and develop adaptable & reusable solutions.

100003/CS622T.3-PO3	**HIGH**	Design solutions for complex engineering problems and perform requirement analysis, identify design methodologies.
100003/CS622T.3-PO4	**HIGH**	Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
100003/CS622T.3-PO5	**HIGH**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools.
100003/CS622T.3-PO6	**MEDIUM**	Perform requirement analysis, identify design methodologies and assess societal, health, safety, legal, and cultural issues.
100003/CS622T.3-PO7	**MEDIUM**	Understand the impact of the professional engineering solutions in societal and environmental contexts and Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions.
100003/CS622T.3-PO8	**HIGH**	Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions by applying ethical principles and commit to professional ethics.
100003/CS622T.3-PO9	**MEDIUM**	Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.
100003/CS622T.3-PO10	**MEDIUM**	Communicate effectively with the engineering community and with society at large to perform requirement analysis, identify design methodologies.
100003/CS622T.3-PO11	**MEDIUM**	Demonstrate knowledge and understanding of engineering requirement analysis by identifying design methodologies.
100003/CS622T.3-PO12	**HIGH**	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change by analysis, identify design methodologies and develop adaptable & reusable solutions.
100003/CS622T.3-PSO3	**MEDIUM**	The ability to apply the fundamentals of computer science in competitive research and prior to that perform requirement analysis, identify design methodologies.
100003/CS622T.4-PO1	**MEDIUM**	Prepare technical report and deliver presentation by applying the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS622T.4-PO2	**HIGH**	Identify, formulate, review research literature, and analyze complex engineering problems by preparing technical report and deliver presentation.

100003/CS622T.4-PO3	**MEDIUM**	Prepare Design solutions for complex engineering problems and create technical report and deliver presentation.
100003/CS622T.4-PO4	**MEDIUM**	Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions and prepare technical report and deliver presentation.
100003/CS622T.4-PO5	**MEDIUM**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools and Prepare technical report and deliver presentation.
100003/CS622T.4-PO8	**HIGH**	Prepare technical report and deliver presentation by applying ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
100003/CS622T.4-PO9	**HIGH**	Prepare technical report and deliver presentation effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.
100003/CS622T.4-PO10	**HIGH**	Communicate effectively with the engineering community and with society at large by prepare technical report and deliver presentation.
100003/CS622T.4-PO11	**MEDIUM**	Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work by prepare technical report and deliver presentation.
100003/CS622T.4-PO12	**HIGH**	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change by prepare technical report and deliver presentation.
100003/CS622T.4-PSO1	**MEDIUM**	Prepare a technical report and deliver presentation to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas.
100003/CS622T.4-PSO2	**MEDIUM**	To acquire programming efficiency by designing algorithms and applying standard practices in software project development and to prepare technical report and deliver presentation.
100003/CS622T.4-PSO3	**MEDIUM**	To apply the fundamentals of computer science in competitive research and to develop innovative products to meet the societal needs by preparing technical report and deliver presentation.
100003/CS622T.5-PO1	**HIGH**	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS622T.5-PO2	**HIGH**	Identify, formulate, review research literature, and analyze complex engineering problems by applying engineering and management principles to achieve the goal of the project.

100003/CS622T.5-PO3	**HIGH**	Apply engineering and management principles to achieve the goal of the project and to design solutions for complex engineering problems and design system components or processes that meet the specified needs.
100003/CS622T.5-PO4	**MEDIUM**	Apply engineering and management principles to achieve the goal of the project and use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
100003/CS622T.5-PO5	**MEDIUM**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO6	**MEDIUM**	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities by applying engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO7	**MEDIUM**	Understand the impact of the professional engineering solutions in societal and environmental contexts, and apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO8	**HIGH**	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice and to use the engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO9	**MEDIUM**	Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO11	**MEDIUM**	Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team. Manage projects in multidisciplinary environments and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO12	**HIGH**	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PSO1	**MEDIUM**	The ability to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas. Apply engineering and management principles to achieve the goal of the project.

100003/CS622T.5-PSO2	**MEDIUM**	The ability to acquire programming efficiency by designing algorithms and applying standard practices in software project development to deliver quality software products meeting the demands of the industry and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PSO3	**MEDIUM**	The ability to apply the fundamentals of computer science in competitive research and to develop innovative products to meet the societal needs thereby evolving as an eminent researcher and entrepreneur and apply engineering and management principles to achieve the goal of the project.