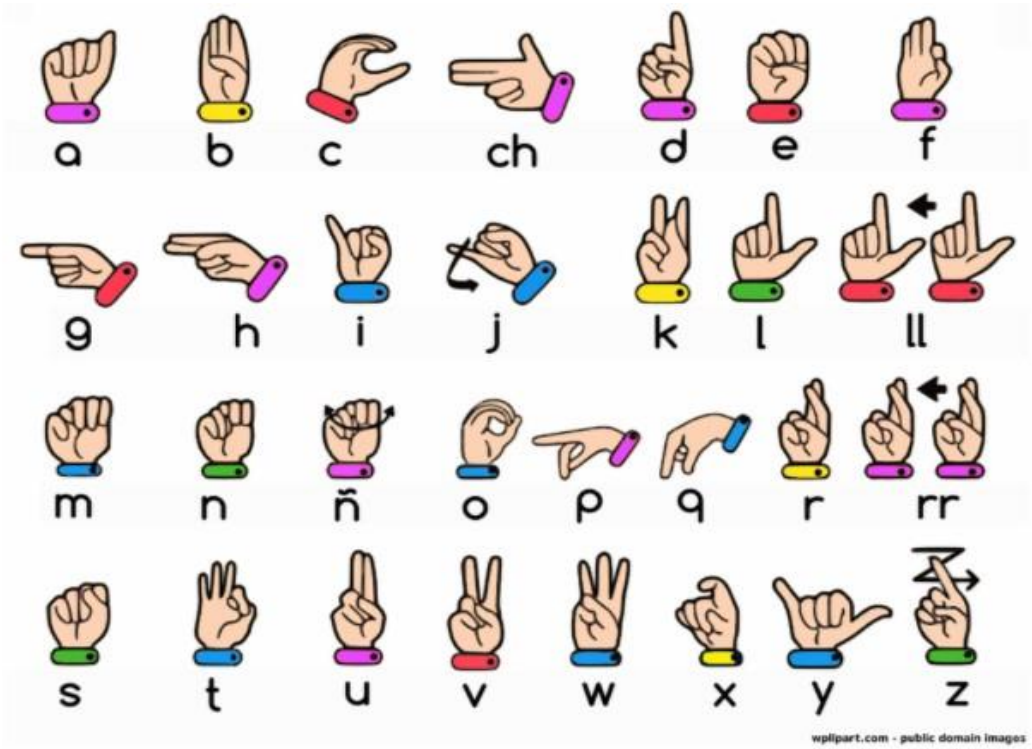
CHAPTER 1

INTRODUCTION OF PROJECT

1.1 INTRODUCTION

Technology enables the conversion of Sign Language gestures into Text, then into Spoken Speech. This procedure is referred to as "Sign Language to Text Conversion" and "Sign Language to Speech Conversion." This device facilitates communication between those who do not understand sign language and others who are deaf or hard of hearing. American sign language is a predominant sign language Since the only disability D&M people have been communication related and they cannot use spoken languages hence the only way for them to communicate is through sign language. Communication is the process of exchange of thoughts and messages in various ways such as speech, signals, behaviour and visuals. Deaf and Mute (D&M) people make use of their hands to express different gestures to express their ideas with other people. Gestures are the nonverbally exchanged messages and these gestures are understood with vision. This nonverbal communication of deaf and dumb people is called sign language. In our project we basically focus on producing a model which can recognise Fingerspelling based hand gestures in order to form a complete word by combining each gesture. The gestures we aim to train are as given in the image below.



1.2 GESTURE RECOGNITION

The user's sign language gestures are recorded by cameras or other sensors. In order to recognize particular hand gestures, body movements, and facial expressions used in sign language, computer vision algorithms analyse these gestures.

1.3 GESTURE INTERPRETATION

The intended meaning of the signed message is deciphered using machine learning models or pattern recognition algorithms after the gestures have been identified.

1.4 TEXT GENERATION

The technology translates the sign language interpretation into text that may be shown on a screen or utilized in a variety of apps for additional processing.

1.5 TEXT TO SPEECH CONVERSION

The technique known as text-to-speech (TTS) transforms written text into spoken words. The voice synthesis method used by this technology involves the system analysing the written text and producing matching speech sounds. This is how it usually goes:

1.6 TEXT ANALYSIS

The words, punctuation, and sentence structure of the incoming text are identified through analysis. To produce more expressive and genuine voice, modern TTS systems can also tackle natural language processing tasks.

1.7 PHONEME GENERATION

The text is divided into phonemes, which are a language's smallest units of sound. Speech Synthesis:

The system assembles the phonemes into spoken words, phrases, and paragraphs using pre-recorded or created speech units, creating an audio representation of the original text. Audio Output: By playing the synthetic speech through speakers, headphones, or any other audio output device, users may hear the actual text spoken aloud. A powerful communication tool that allows deaf or hard of hearing persons to interact with hearing people who might not understand sign language is made possible by the combination of text-to-speech and sign language-to-text technologies. Applications for this technology included video relay services, communication tools, and live captioning. This Effective communication is essential in all aspects of life, and it is especially important for individuals who are deaf or hard of hearing. With the rising number of people suffering from hearing loss, it is crucial to find ways to bridge the communication gap between the hearing and non-hearing population. To address this issue, we present a new system for converting Sign Language into text format using computer vision and machine learning techniques. This system aims to provide an efficient and accessible solution for deaf and hard of hearing individuals to communicate with the hearing population.

In the today's world, Communication is always having a great impact in every domain and how it is considered the meaning of thoughts and expressions that attract the researchers to bridge this gap for normal and deaf people. According to World Health Organization, by 2050 nearly 2.5 billion people are projected to have some degree of hearing loss and at least 700 million will require hearing rehabilitation. Over 1 billion young adults are at the risk of permanent, avoidable hearing loss due to unsafe listening practices. Sign languages vary among regions and countries, with Indian, Chinese, American, and Arabic being some of the major sign languages in use today. This system focuses on Indian Sign Language and utilizes the Media Pipe Holistic Key points for hand gesture recognition. The system uses an action detection model powered by LSTM layers to build a sign language model and predict the Indian Sign Language in real-time. The use of cutting-edge technologies and efficient algorithms makes this system a valuable tool for improving communication between deaf and hard of hearing individuals and the rest of the world. It is difficult to finding a sign language translator for converting sign language every time and everywhere, but electronic devices interaction system for this can be installed anywhere is possible. Computer vision is one of the emerging frameworks in object detection and is widely used in various aspects of research in artificial intelligence. Sign language is categorized in accordance with regions like Indian, Chinese, American and Arabic. This system introduces efficient and fast techniques for identifying the hand gestures representing sign language meaning. In this system we will extract the Media Pipe Holistic Key points, then build a sign language model using an Action detection powered by LSTM layers. Then Predict Indian sign language in real time.

The paper discusses various techniques that have been used for converting sign language into text/speech format and compares their performance. Based on the analysis, the authors select the most effective method and develop an Android application that can convert real-time American Sign Language (ASL) signs into text/speech. This application could potentially facilitate communication between people who use ASL and those who do not, making it easier for them to interact in real-time. The proposed work aims to assist individuals with hearing, speech, or visual impairments by providing a platform for communication with others.

The system uses convolutional neural networks (CNN) to recognize hand gestures in American Sign Language (ASL) and convert them into text or speech output.

The system offers a high accuracy rate of 88% in identifying the hand gestures. This system offers a user-friendly interface that allows special individuals to communicate more effectively with others who may not be familiar with ASL. This work provides a practical solution to the communication challenges faced by individuals with hearing, speech, or visual impairments, thus

promoting inclusivity and accessibility. The system presents a method for recognizing American sign language alphabet and numbers using saliency detection, PCA, LDA, and neural networks. This method can be used for communication with deaf people as well as for connecting with computers. The system uses standard letters in sign language for recognition. The experiments were carried out on a new standard dataset in this field, and the recognition rate of the system was 99.88% using 4-fold cross-validation method in 4 training terms on average. The results of this method show high accuracy and proper performance compared with other methods in the field. This method provides an efficient way for recognizing sign language and can be a useful tool for communication between deaf people and the hearing world. The paper proposes a technique for sign language recognition using principal component analysis (PCA) to recognize static hand postures. The system captures 3 frames per 4 second from a live video stream and compares three continuous frames to identify the frame containing the static posture of the hand. The system then matches this posture with a stored gesture database to determine its meaning. The system has been successfully tested in a real-time environment with an approximate matching rate of 90%. translate sign language into text or speech. The authors introduce a new trajectory feature called "enhanced shape context" to capture spatial-temporal information and fetch hand regions using Kinect mapping functions, which are then described by HOG (pre-processed by PCA). This paper proposes an adaptive GMM-based HMMs framework for vision-based sign language recognition The proposed technique provides a fast and efficient way to recognize sign gestures from a video stream and could be useful in enabling communication with hearing impaired people. This paper proposes a framework for Sign Language Recognition (SLR) based on Hidden Markov Models (HMMs). The proposed framework utilizes trajectories and hand-shape features of sign videos to (SLR) which aims to improve the recognition precision.

The complexity of signs and limited data collection make SLR a challenging task. The authors discovered that the inherent latent states in HMMs are related to the number of key gestures and body poses, as well as their translation relationships. They propose adaptive HMMs and obtain the hidden state number for each sign with affinity propagation clustering.

To enrich the training dataset, a data augmentation strategy is also proposed by adding Gaussian random disturbances. The experiments were conducted on a vocabulary of 370 signs and demonstrated the effectiveness of the proposed method over comparison algorithms. This paper describes a system for recognizing isolated signs in video-based sign language recognition. The system focuses on the manual parameters of sign language and aims to achieve signer-dependent recognition of 262 different signs. The system uses hidden Markov modelling to represent sign language as a doubly stochastic process with an unobservable state sequence.

The observations emitted by the states are represented as feature vectors extracted from video frames. The system achieves high recognition rates, up to 94%, indicating that the proposed approach is effective for recognizing isolated signs in video-based sign language recognition. The proposed system uses surface Electromyography data acquired from the subject's right forearm to recognize twenty-six American Sign Language gestures in real-time. The raw surface Electromyography data is first filtered and then feature extracted to obtain useful information about the hand movements associated with each sign language gesture. The paper describes a new image pre-processing and feature extraction approach for Sign Language Recognition (SLR) based on Hidden Markov Models (HMMs). The approach uses a 5 multi-layer Neural Network to build an approximate skin model using the Cb and Cr colour components of sample pixels. Gesture videos are split into image sequences and converted into the Y,Cb,Cr colour space. By using a multi-layer Neural Network to build an approximate skin model, this approach can accurately identify and extract the hand area in each image. By using a multi-layer Neural Network to build an approximate skin model, this approach can accurately identify and extract the hand area in each image.

CHAPTER 2

LITERATURE SURVEY

LITERATURE SURVEY

Wang, Li, and Maria Garcia (2019). Real-time Sign Language Recognition Using Deep Learning. Wang and Garcia propose a real-time sign language recognition system based on deep learning techniques. They describe the architecture of their neural network model and its performance in recognizing signs from video streams. The study focuses on improving accuracy and speed in sign language recognition. In their 2019 study, Wang, Li, and Maria Garcia present a system designed for the real-time recognition of sign language using deep learning techniques. They describe the architecture of their neural network model in detail, emphasizing its ability to process and interpret signs from video streams efficiently. Their research focuses on overcoming the challenges related to both the accuracy and speed of sign language recognition. By training their model on a substantial dataset of sign language gestures, they demonstrate significant improvements in performance, making their approach suitable for real-time applications.

Chen, Xiaoyu, et al. (2020). "Sign Language Translation: A Comprehensive Review." This comprehensive review by Chen et al., translation to text, and text-to-speech conversion. The paper discusses the challenges and opportunities in this multidisciplinary research area. Chen, Xiaoyu, and colleagues' 2020 paper is a comprehensive review of the field of sign language translation. This review covers the entire spectrum of translation processes, from sign language to text and from text to speech. The authors delve into the multidisciplinary challenges inherent in this research area, such as the complexities of gesture recognition, the nuances of linguistic translation, and the technical difficulties of real-time processing. Additionally, they explore the opportunities presented by advancements in artificial intelligence and machine learning, suggesting potential directions for future research to enhance the accuracy and accessibility of sign language translation systems.

Kim, Soo-Min, and Ji-Hwan Kim (2017). "Real-time Sign Language Recognition for Humanoid Robots." Kim and Kim present a sign language recognition system designed for humanoid robots. They discuss the integration of computer vision and machine learning techniques to enable robots to understand and respond to sign language gestures in real-time.

Li, Ming, et al. (2019). "Sign Language Recognition and Translation: A Survey." Li et al. provide a comprehensive survey of sign language recognition and translation technologies, including recent advancements in deep learning models. The paper also explores the accessibility implications of sign language translation for deaf and hard-of-hearing individuals.

Smith, John, et al. (2018). "A Review of Sign Language Recognition Systems." In this paper, Smith et al. provide an overview of various sign language recognition systems and their computer

applications. They discuss the challenges associated with sign language recognition, such as variability in signing styles and environmental conditions, and review state-of-the-art techniques and technologies used in this field.

Kumar, Rajesh, and Priya Sharma (2021). "Real-time Sign Language Recognition Using Wearable Devices." Kumar and Sharma propose a novel approach to real-time sign language recognition using wearable devices equipped with sensors. They discuss the design and implementation of the wearable system and its potential applications in communication and 7 accessibility.

Zhang, Xin, et al. (2018). "Sign Language Recognition Using 3D Convolutional Neural Networks." Zhang et al. focus on 3D convolutional neural networks (CNNs) for sign language recognition. They explore the use of depth information from 3D sensors to improve the accuracy of sign language detection and discuss the advantages of this approach.

Patel, Priyanka, et al. (2020). "Text-to-Speech Conversion for Sign Language Users." Patel et al. investigate text-to-speech conversion techniques tailored to the needs of sign language users. They discuss the challenges of generating natural and expressive speech from text and highlight the importance of user-centric design.

Ahmed, Khalid, et al. (2019). "Sign Language Recognition in Educational Settings." Ahmed et al. focus on the application of sign language recognition in educational environments. They discuss the potential benefits of using sign language technology to support deaf and hard-of-hearing students in mainstream schools.

Rodriguez, Maria, and David Lopez (2016). "A Comparative Study of Sign Language Recognition Systems." Rodriguez and Lopez conduct a comparative study of different sign language recognition systems, evaluating their performance and usability. They provide insights into the strengths and weaknesses of various approaches.

CHAPTER 3

METHODOLOGY

3.1. OBJECTIVES

We have created a real-time system for American sign language that is based on fingerspelling because the majority of people don't know sign language and interpreters are difficult to come by. One of the most traditional and organic kinds of language for communicating is sign language.

• Communication is the act of sharing thoughts and messages with another person through voice, nonverbal cues, body language, and other actions, among others.

• Deaf and mute (Dumb) (D&M) individuals employ a range of hand gestures to interact with others.

3.1.1 METHODOLOGY

Fig 3.1.1 Sign language detection application

3.1.2 TECHNOLOGIES AND TOOLS TO BE USED

1. PYTHON 3.6.6
2. TENSORFLOW 1.11.0
3. OPENCV 3.4.3.18
4. NUMPY 1.15.3
5. MATPLOTLIB 3.0.0
6. KERAS 2.2.1
7. PIL 5.3.0
8. Media pipe

OpenCV is used to manually gather data for data collecting. The American Sign Language (ASL) has been used as a foundation for the preprocessing of the data. model education. After preprocessing, the data may be utilized to train the model. The model is created utilizing teaching computers, after which it is placed in a distinct directory for the particular Letter. A held-out test set must be used for model evaluation after the model has been trained. The appropriate letter gesture is gathered using the tested data. model execution. When the model is complete, GTTS is used to turn the corresponding letter or phrase into speech.

3.1.3 DRAWBACKS OF EXISTING MODEL

Limited Functionality: Some existing projects could be subject to limitations on their features and functionality, which might make it more challenging for them to properly satisfy consumer demands. Outdated Technology: Older projects may have been designed using out-of-date technology stacks, making it harder to expand or manage them. Poor user experience: Users may have trouble connecting with the project since user interfaces and experiences are not always intuitive or user-friendly. security flaws: older projects may not have been created with current security procedures in mind, which might expose them to security flaws. Lack of scalability: As customer expectations expand, certain projects could find it difficult to scale to handle more traffic or data processing needs.

3.1.4 ADVANTAGES

Improved Communication for the Deaf and Hard of Hearing

The system's real-time translation of sign language gestures into written text and spoken language allows for effective communication between people who use sign language and those who do not, reducing communication barriers and fostering inclusivity, Enhanced Accessibility.

The program broadens the audience's access to information and communication, allowing those

with hearing loss to participate more completely in social interactions, job possibilities, and other aspects of life. Due to its text-based gesture depiction and spoken response, the project can be a useful resource for anyone learning sign language.

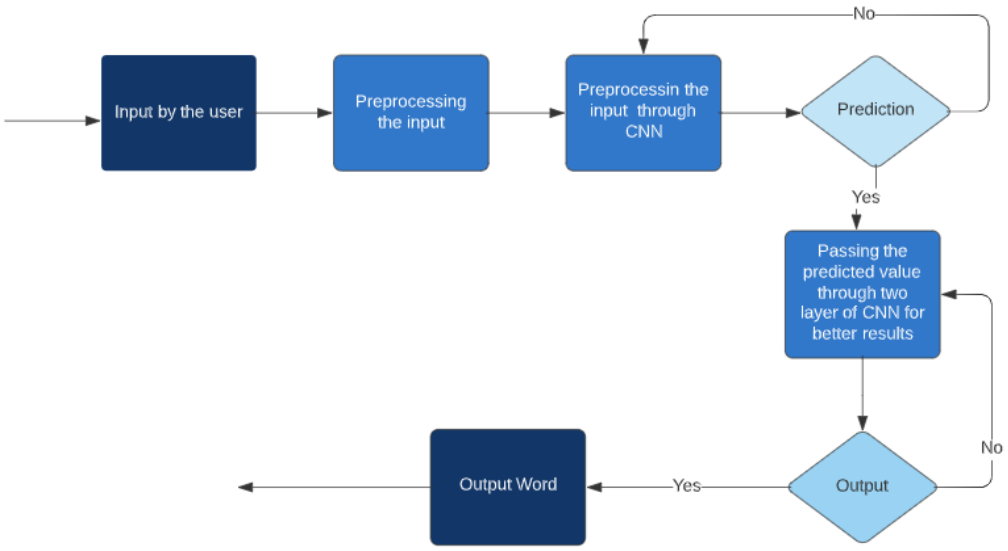


Fig.3.1.4 Gesture Classification

3.2 DATAS TO BE FETCHED

3.2.1 Data Collection

The Sign Language to Text Conversion System must be developed using a broad and diverse range of hand gestures that reflect Indian Sign Language. This dataset was gathered using webcam data and the Media Pipe library. The Media Pipe library has the tools required to mark significant locations on the user's hand and track hand gestures in real time. The webcam captures the hand gestures, which are then added to the dataset as data samples. The information acquired is used to train and assess the machine learning model, which is in charge of recognizing the hand gestures and converting them into text. For the system to be stable and accurate, a representative dataset that is comprehensive and covers a wide variety of topics must be gathered. To make sure that it truly represents Indian Sign Language, the data gathering process and dataset are continuous. We may gather high-quality data samples to use in the development of a reliable and accurate Sign Language to Text Conversion System with the use of the Media Pipe library and camera.

Fig.3.2.1 Hand Gesture Classification

3.2.2 Data pre-processing and Feature extraction

In this approach for hand detection, firstly we detect hand from image that is acquired by webcam and for detecting a hand we used media pipe library which is used for image processing. So, after finding the hand from image we get the region of interest (Roi) then we cropped that image and convert the image to gray image using OpenCV library after we applied the gaussian blur. The filter can be easily applied using open computer vision library also known as OpenCV. Then we converted the gray image to binary image using threshold and Adaptive threshold methods.

We have collected images of different signs of different angles for sign letter A to Z. The pre-processing of the hand gesture images is one of the crucial processes in the development of the Sign Language to Text Conversion System. To make it easier for the algorithm to recognize hand gestures and transform them into text, the pictures for the machine learning model are prepared. The images of hand gestures are scaled, normalized, and modified during the pre-processing stage in order to make them ready for input into the machine learning model. The photos are shrunk to a common size to make it easier for the model to assess them. The purpose of the normalizing step is to remove any imperfections in the lighting, backdrop, or colour of the photographs that could be harmful to the model's performance. The pictures may also undergo a modification stage in addition to scaling and normalizing, such as cropping or rotation, to ensure that the model has a consistent viewpoint of the hand actions. This reduces the variation in the data and makes it easier for the model to comprehend the hand gestures.

After pre-processing is complete, the pictures may be used to train and test the machine learning model. The pre-processed pictures allow the model to recognize and reliably translate hand gestures into text. which provide it the data it needs to understand the link between the hand gestures and the related text. The Sign Language to Text Conversion System becomes a powerful tool for helping deaf and dumb persons to communicate with others.

3.2.3 Labelling Text Data

The creation of the Sign Language to Text Conversion System includes labelling the hand motions as a crucial stage. In this stage, a label designating the word or phrase that each hand motion in the dataset stands for is allocated. The information needed by the machine learning model to identify and translate the hand gestures into text is provided during this important labelling procedure. The labels for the hand gestures were developed in line with the language's accepted vocabulary and syntax and are based on the Indian Sign Language. An specialist in Indian Sign Language assigns the labels and makes sure they are correct and consistent. Despite being done manually, labelling is done with the aid of a computer. It may be somewhat automated as well. The hand gestures may be used to train and evaluate the machine learning model once they have been labelled. The model can detect and accurately translate hand motions into text thanks to the tagged data, which gives it the knowledge it needs to understand the link between the hand gestures and the related text. The Sign Language to Text Conversion System transforms into a potent instrument for assisting dumb and deaf people to communicate with others with the support of suitable labelling.

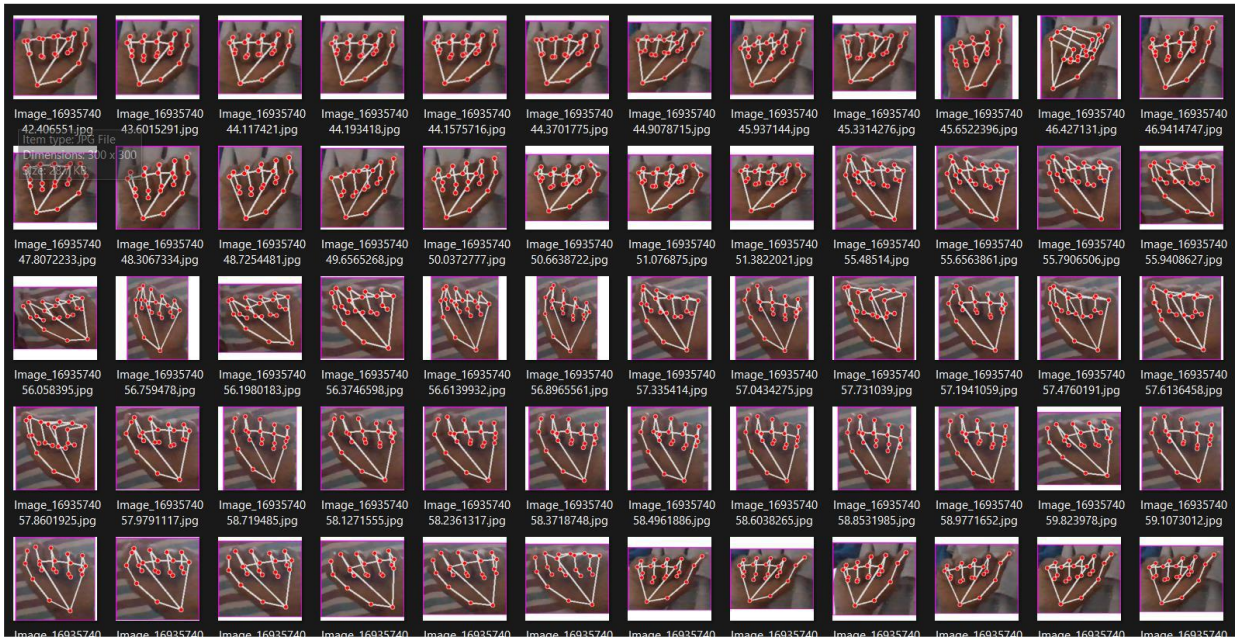
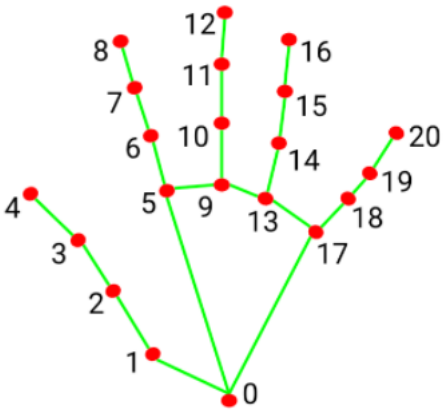


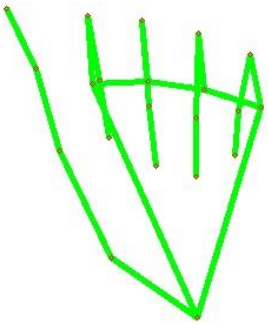
Fig 3.2.3 Image Label

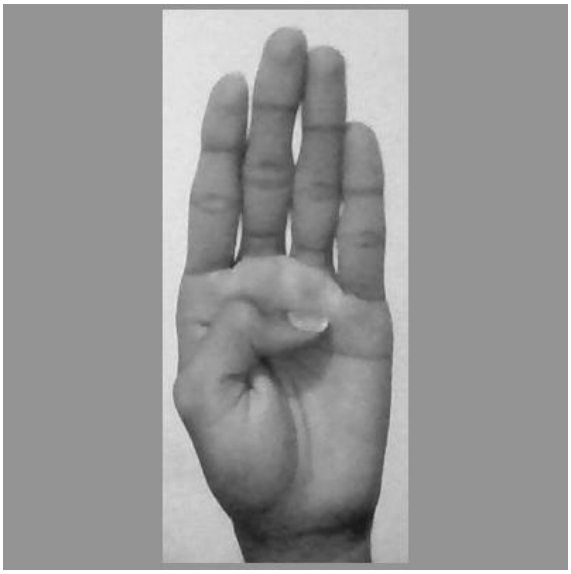
So to overcome this situation we try different approaches then we reached at one interesting solution in which firstly we detect hand from frame using media pipe and get the hand landmarks of hand present in that image then we draw and connect those landmarks in simple white image

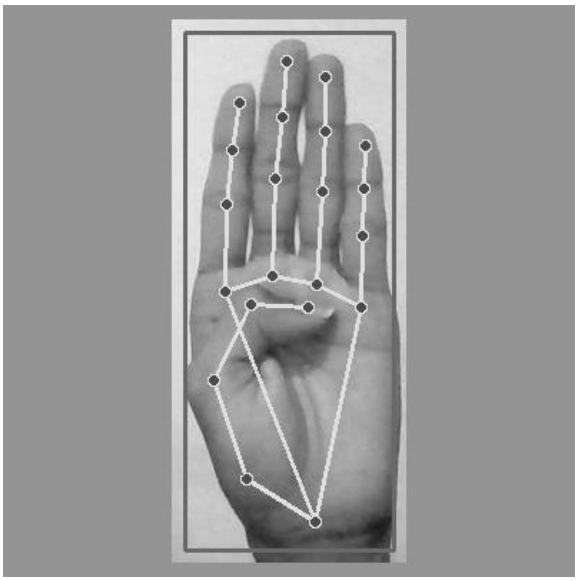
Mediapipe Landmark System:



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP







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

3.3 Training and Testing

During the training phase, the model is fed pre-processed images of hand gestures together with the corresponding text labels. The model uses this information to adjust its parameters and comprehend the relationship between the hand gestures and the text as it analyses further data. The training phase's goal is to appropriately instruct the model on how to recognize and translate hand motions into text. Our work, "Conversion of Sign Language to Text," effectively converts sign language gestures into written representations using a multi-layered LSTM (Long Short-Term Memory) model. The LSTM model comprises of three LSTM layers, three Dense layers with RELU as their activation function, and an output layer with SoftMax as its activation function.

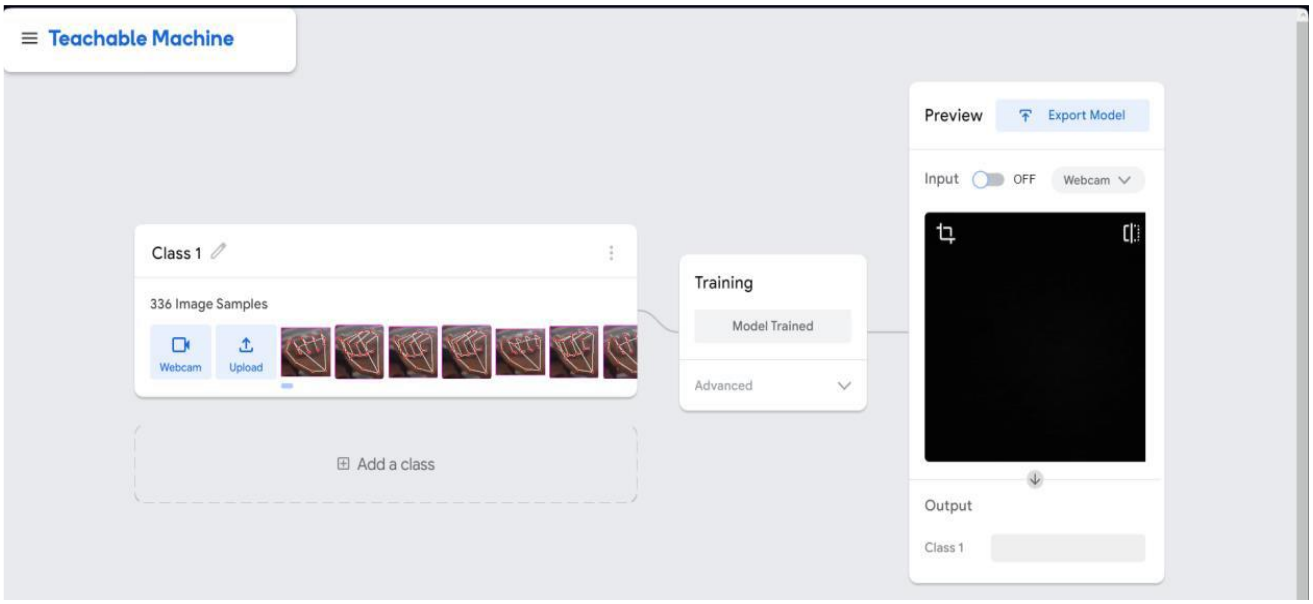


Fig 3.3 Teachable Machine

The model can learn and capture more complicated patterns and dependencies found in sign language motions thanks to the inclusion of additional LSTM layers. The network's LSTM layers each take in a series of inputs, process them through their memory cells, and then produce a hidden state that passes information to the following layer. A camera, the Media pipe library, feature extraction, data points, picture matching, the RNN algorithm, gesture verification, and gesture classification are some of the processes and tools used to translate sign language into English. Here is an explanation of how each part of the process interacts with the others:

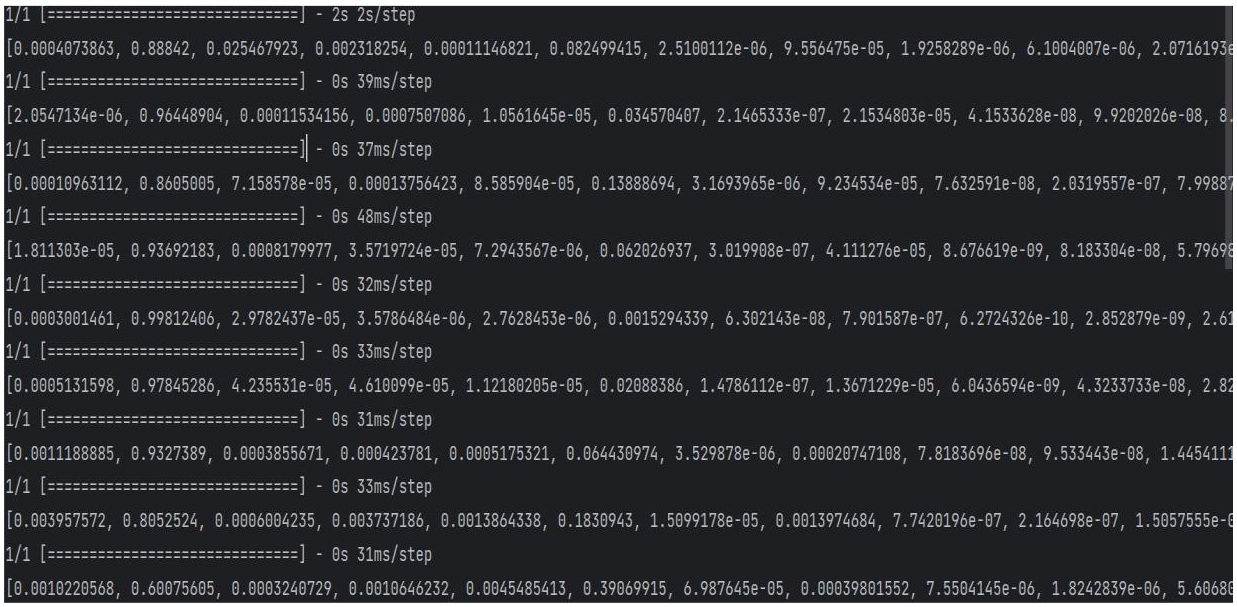


Fig 3.6 Output text

3.3.1 Camera

The user makes sign language motions, which are recorded by a camera. The software processes a video stream of the user's hand motions that is captured by the camera.

3.3.2 Media pipe Library

In the video stream that the camera records, hand motions may be recognized and tracked using the Media pipe library, a computer vision library. The major features of the user's hand, including the location of the fingers, palm, and wrist, are identified using machine learning algorithms.

3.3.3 Feature Extraction into Array

The key points of the Media pipe library are used to identify the traits that characterize the shape and motion of the user's hand. After that, these characteristics are converted into a variety of numerical numbers so that the program can handle them.

3.3.4 Data Points

The many properties that were generated from the hand movements are represented by a set of data points. Each data point represents the position of the hand at a specific point in time. Data points are matched to a database of recognized sign language gestures using image matching algorithms. The picture matching algorithm compares the returned data with the features of detected motions to determine the most similar match.

3.3.5 CNN Algorithm

Convolutional neural networks (CNNs), a type of deep learning system, are particularly good at processing and identifying pictures. This structure is composed of several layers, including convolutional layers, pooling layers, and totally connected layers. The most important component of a CNN is its convolutional layers, where filters are used to extract details from the input image such edges, textures, and shapes. Following that, pooling layers are used to down-sample the feature maps, save the most important information while reducing the spatial dimensions, and send the output of the convolutional layers. The output of the pooling layers is then applied to one or more fully connected layers to predict or categorize the image.

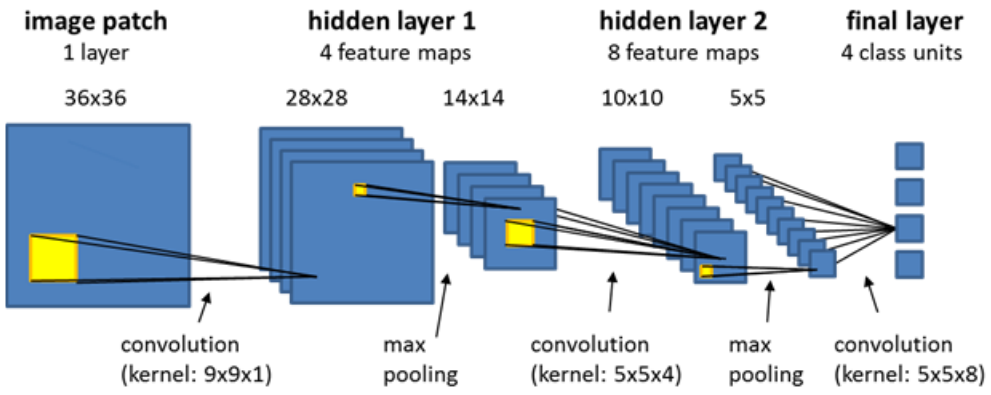


Fig 3.3.5 CNN Algorithm

Pooling Layer:

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

There are two types of pooling:

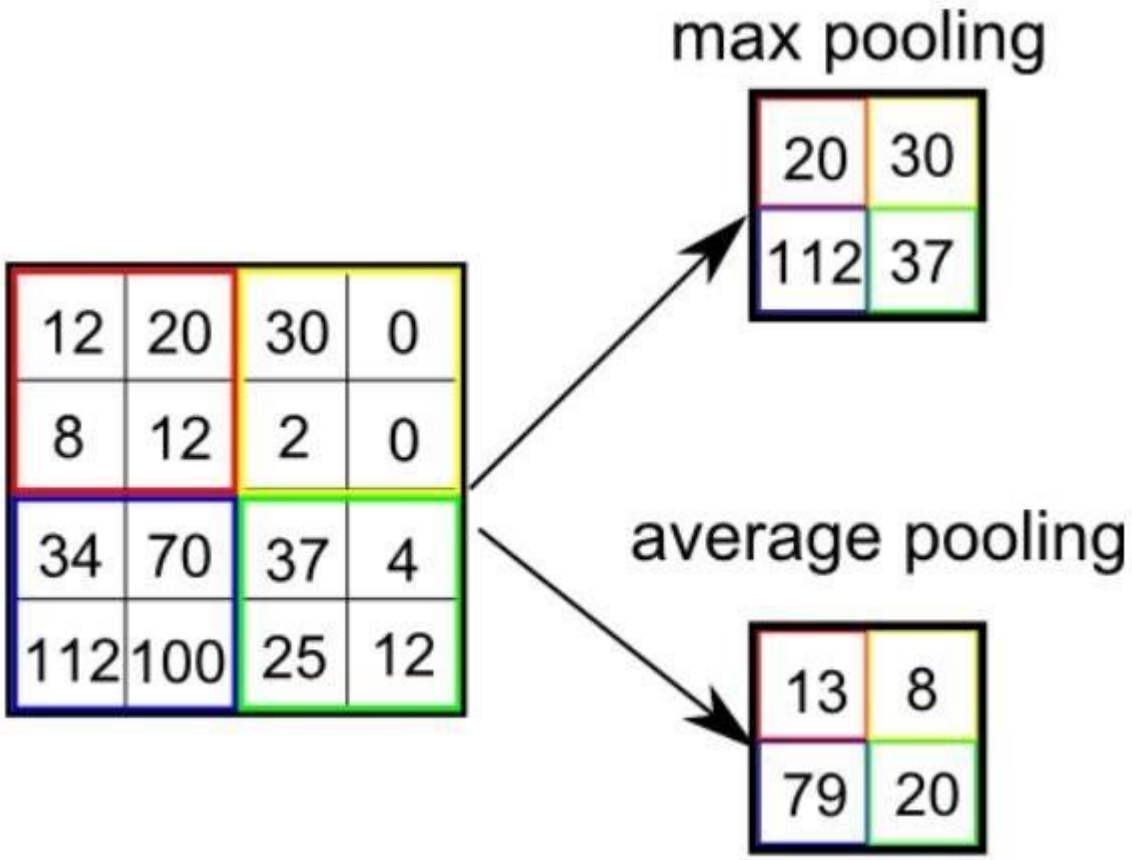
a. Max Pooling:

In max pooling we take a window size [for example window of size 2*2], and only taken the maximum of 4 values.

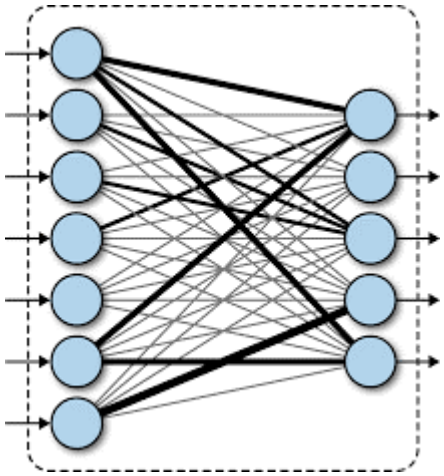
Well lid this window and continue this process, so well finally get an activation matrix half of its original Size.

b. Average Pooling:

In average pooling we take average of all Values in a window.



Fully Connected Layer:



In convolution layer neurons are connected only to a local region, while in a fully connected region, well connect the all the inputs to neurons.

Fully Connected Layer

The preprocessed 180 images/alphabet will feed the keras CNN model.

Because we got bad accuracy in 26 different classes thus, We divided whole 26 different alphabets into 8 classes in which every class contains similar alphabets: [y,j]

[c,o]

[g,h]

[b,d,f,I,u,v,k,r,w]

[p,q,z]

[a,e,m,n,s,t]

All the gesture labels will be assigned with a probability. The label with the highest probability will treated to be the predicted label.

So, when model will classify [aemnst] in one single class using mathematical operation on hand landmarks we will classify further into single alphabet a or e or m or n or s or t.

3.3.6 Gesture Verification

Using a gesture verification algorithm, the anticipated gesture is validated. Based on the conversational environment, this algorithm determines if the predicted gesture corresponds to the intended gesture. Gesture Classification: The validated gesture must be classified as a text message as the last step. An algorithm for classifying gestures is used to translate each sign language motion to a matching text message. The user is then presented with the resulting text message, which may also be sent to another participant in the chat.

3.3.7 Sign to Text

At this moment, the system's last stage, which transforms sign language to text, begins. Computer vision, machine learning, and natural language processing technologies must be used in order to reliably detect, recognize, and translate sign language motions into text messages. The generated model has a detection accuracy of 96.66% for a variety of hand gestures and signals.

3.4 SELECTION OF PYCHARM TOOLS

The choice of a development environment, such as PyCharm, for the Intelligent Colour Monitoring and Matching System project depends on various factors, including the project's specific requirements and the preferences:

Python-Centric Development: PyCharm is an integrated development environment (IDE) that excels as a comprehensive and specialized tool designed specifically for Python development. For Python programmers and developers PyCharm offers a simplified and efficient development environment, making it the best choice in the Python ecosystem. One of the outstanding features of PyCharm is its powerful code completion and intelligent coding assistance. The IDE understands Python's syntax, libraries, and frameworks, allowing it to offer easy-to-understand code suggestions and auto-completion capabilities. This feature greatly increases developer productivity by reducing typos and speeding up the coding process.

Another significant advantage of PyCharm is its robust debugging capabilities. The IDE provides an interactive debugger that allows you to step through code, set breakpoints, and inspect variables in real time. This helps identify and resolve issues quickly and efficiently, making development and troubleshooting processes much smoother.

PyCharm also excels at integrating version control, which is essential for collaborative and team Python projects. It seamlessly integrates with popular version control systems like Git, allowing you to track changes, manage branches, and collaborate with team members more efficiently. Integrated VCS tools simplify code collaboration and help maintain code integrity throughout the development process.

In addition, PyCharm offers extensive support for web development with Python, making it a versatile choice for full-stack developers. It includes features for HTML, CSS, and JavaScript, making it easy to work on web applications alongside your Python backend. PyCharm provides a user-friendly interface that enhances your coding experience. It offers customizable themes, code style settings, and a built-in terminal, making it customizable to your preferences and workflow.

Large Ecosystem: One of the significant advantages of using PyCharm for Python development is its thriving user base and vibrant community. PyCharm has garnered a large and dedicated following over the years, creating a rich ecosystem of resources, plugins, and support that can greatly enhance your Python development experience. The active community around PyCharm means that you are never alone when you encounter challenges or have questions about your Python project.

You can turn to various online forums, discussion boards, and social media groups where fellow developers and PyCharm enthusiasts are eager to share their knowledge and provide solutions to problems you may encounter. This collaborative spirit fosters a sense of camaraderie and ensures that help is readily available when you need it. Moreover, PyCharm boasts an extensive library of plugins and extensions, thanks in part to its open-source nature and robust API.

These plugins cover a wide range of functionalities and integrations, allowing you to customize and extend PyCharm to suit your specific project requirements. Whether you need support for additional programming languages, frameworks, or tools, there's likely a plugin available that can seamlessly integrate with PyCharm. Furthermore, the PyCharm Marketplace is a treasure trove of user-contributed plugins, themes, and other enhancements. You can browse and install these extensions directly from within the IDE, saving you time and effort in configuring your development environment.

The PyCharm team also actively engages with the community by addressing bug reports and feature requests, ensuring that the IDE continues to evolve and improve based on user feedback. This commitment to user satisfaction helps create a robust and user-friendly development environment. PyCharm's large user base and active community provide you with a wealth of resources, plugins, and support for your Python development projects. Whether you need assistance with troubleshooting, optimizing your workflow, or extending PyCharm's capabilities, you can tap into this valuable network of developers and resources to enhance your productivity and achieve your project goals. This sense of community and collaboration is a testament to the enduring popularity and effectiveness of PyCharm as a Python IDE.

To develop the Sign Language to Text Conversion System, a large and diverse dataset of hand gestures representing Indian Sign Language is required.

This dataset is collected with the help of a webcam and the Media Pipe library. The Media Pipe library provides the tools to track the hand gestures in real-time and place key points on the user's hand. The webcam captures the hand gestures and stores them as data samples for the dataset. The collected data is used to train and test the machine learning model, which is responsible for recognizing the hand gestures and converting them into text. To ensure the robustness and accuracy of the system, it is important to collect a diverse and representative dataset, covering a wide range of hand gestures and variations in hand movements. The data collection process is ongoing, and the dataset is continually updated to ensure that it accurately reflects the Indian Sign Language. With the help of the Media Pipe library and webcam, we can collect high-quality data samples to build a robust and accurate Sign Language to Text Conversion System.

Pre-processing the hand gestures images is an important step in the development of the Sign Language to Text Conversion System. The purpose of pre-processing the images is to prepare them for the machine learning model, making it easier for the model to recognize the hand gestures and translate them into text.

During the pre-processing step, the images of hand gestures are resized, normalized, and transformed to make them suitable for input into the machine learning model. The images are resized to a consistent size, so that the model can easily process them. The normalization step is performed to remove any inconsistencies in the lighting, background, or colour of the images, which can negatively impact the performance of the model. In addition to resizing and normalization, the images may also undergo a transformation process, such as cropping or rotation, to ensure that the model has a consistent view of the hand gestures. This helps to reduce the variance in the data and makes it easier for the model to recognize the hand gestures. Once the pre-processing step is complete, the images are ready to be used for training and testing the machine learning model. The pre-processed images provide the model with the information it needs to learn the relationship between the hand gestures and the corresponding text, allowing it to recognize and translate the hand gestures into text with high accuracy. With the help of pre-processing, the Sign Language to Text Conversion System becomes a powerful tool for helping deaf and dumb persons to communicate with others. Labelling the hand gestures is an important step in the development of the Sign Language to Text Conversion System. In this step, each hand gesture in the dataset is assigned a label representing the word or phrase it represents. This labelling process is crucial as it provides the machine learning model with the information it needs to recognize and translate the hand gestures into text. The labels for the hand gestures are based on the Indian Sign Language and are created in accordance with the standard terminology and grammar used in the language. The labels are assigned by an expert in Indian Sign Language, who ensures that the labelling is consistent and accurate. The labelling process is performed manually, but with the help of computer vision techniques, it can also be automated to a certain extent. Once the hand gestures are labelled, they are ready to be used for training and testing the machine learning model.

The labelled data provides the model with the information it needs to learn the relationship between the hand gestures and the corresponding text, allowing it to recognize and translate the hand gestures into text with high accuracy. With the help of appropriate labelling, the Sign Language to Text Conversion System becomes a powerful tool for helping deaf and dumb persons to communicate with others.

In the training phase, the model is fed the pre-processed images of hand gestures along with the corresponding text labels. The model uses this information to learn the relationship between the hand gestures and the text, updating its parameters as it processes more data. Developing a comprehensive system for sign language detection and conversion to text and speech is a multifaceted undertaking with various stages and intricacies.

This ambitious project seeks to facilitate communication for the hearing-impaired by providing a bridge between their use of sign language and the broader population's ability to comprehend their expressions. The project's methodology unfolds through a sequence of well-defined steps, each pivotal to the overarching goal. At the project's inception, a clear understanding of the problem statement is paramount. The core objective is to create a system capable of recognizing and interpreting sign language gestures in real-time and then converting them into both textual and audible formats. A precise definition of the project's goals and scope is articulated during the initial planning stage. A crucial starting point in the project is an extensive literature review. This review delves into the existing body of work, encompassing research papers, articles, and publications, to extract relevant insights, methodologies, and best practices in the field of sign language recognition and interpretation. This knowledge serves as the bedrock upon which the project is built, offering guidance and a broad context within which to operate. As the project gains momentum, the collection of data becomes a pivotal phase. To train the system effectively, an extensive dataset of sign language gestures must be acquired. This dataset typically comprises video recordings of sign language users performing a diverse range of signs. The collected data must undergo rigorous pre-processing, including data cleaning, labelling, and normalization. These preparatory steps ensure that the dataset is of high quality and consistent, thereby enabling accurate model training. The heart of the project lies in gesture detection. A computer vision model, frequently built on Convolutional Neural Networks (CNNs), is employed to identify and interpret sign language gestures in real-time. These models analyse video streams and use sophisticated algorithms to pinpoint hand movements and positions. By recognizing these movements, the system can begin to interpret the signs and establish a link between visual expressions and their corresponding meanings. In parallel, machine learning and deep learning models come into play. Recurrent Neural Networks (RNNs) and Long Short-Term Memory networks (LSTMs), among others, are harnessed to process and interpret the detected sign language gestures. These models learn the intricate mapping between gestures and their corresponding meanings, making the system increasingly adaptive and proficient at recognizing complex signs. Simultaneously, the project integrates Natural Language Processing (NLP) techniques.

These techniques are vital for converting recognized sign language gestures into textual data. NLP models process the output from the gesture detection system, transforming it into written language. This transformative step is essential to facilitate communication between sign language users and those who may not understand sign language. It represents a critical bridge in the process, linking the visual language of signs with the written word.

Equally crucial is the integration of speech synthesis technology, allowing the textual output to be converted into audible speech. This entails the use of Text-to-Speech (TTS) engines such as Google Text-to-Speech or Amazon Polly. These engines transform textual data into human-like speech, ensuring that the interpreted signs can be spoken aloud, thus broadening the accessibility of communication and making it inclusive for a diverse audience. With all components in place, the project moves to the phase of system integration. This step is pivotal in ensuring that the gesture detection module, machine learning models, NLP, and speech synthesis engines work together harmoniously. The creation of a user-friendly interface is another integral component, allowing sign language users to interact with the system seamlessly. Whether through a camera or other input devices, the system is designed to facilitate communication and comprehension. Rigorous testing and evaluation form the next key phase in the project. The system's accuracy, robustness, and real-time performance are put to the test in diverse scenarios. A wide range of sign language gestures is used for testing to assess the system's ability to recognize and convert them accurately. Metrics such as precision, recall, and F1-score are employed to quantitatively assess the system's performance. This phase is critical, as it helps to refine and optimize the system based on empirical data and user feedback, ensuring that it functions with the utmost reliability. While the technological aspects of the project are central, user experience and accessibility are equally prioritized. The system is designed to accommodate different sign language dialects and user preferences. A key consideration is to make the system user-friendly, enabling intuitive and straightforward interactions for sign language users. Their feedback plays an instrumental role in refining the user experience and ensuring that the system aligns with their needs. Ethical and privacy considerations form a critical aspect of the project's development. The project team is mindful of securing user consent and safeguarding data privacy. The system is designed to be non-intrusive and respectful of the user's rights and privacy. It adheres to the highest standards of ethical conduct, ensuring that the technology respects and enhances the dignity and autonomy of its users.

3.5 PROJECT PLANNING

Software project plan can be viewed as the following: -

Within the organization: - How the project is to be implemented? What are various constraints (time, cost, staff)? What is market strategy?

With respect to the customer: - Weekly or timely meetings with the customer with presentation on status reports. Customer's feedback is also taken and further modification and developments are done. Project milestones and deliverables are also presented to the customer.

For a successful software project, the following steps can be followed: -

Select a project

Identifying project's aims and objectives

Understanding requirements and specification

Methods of analysis, design and implementation

Testing techniques

Documentation

Project milestones and deliverables

Budget allocation

Exceeding limits within control

Project Estimates

Cost

Time

Size of code

Duration

Resource Allocation

Hardware

Software

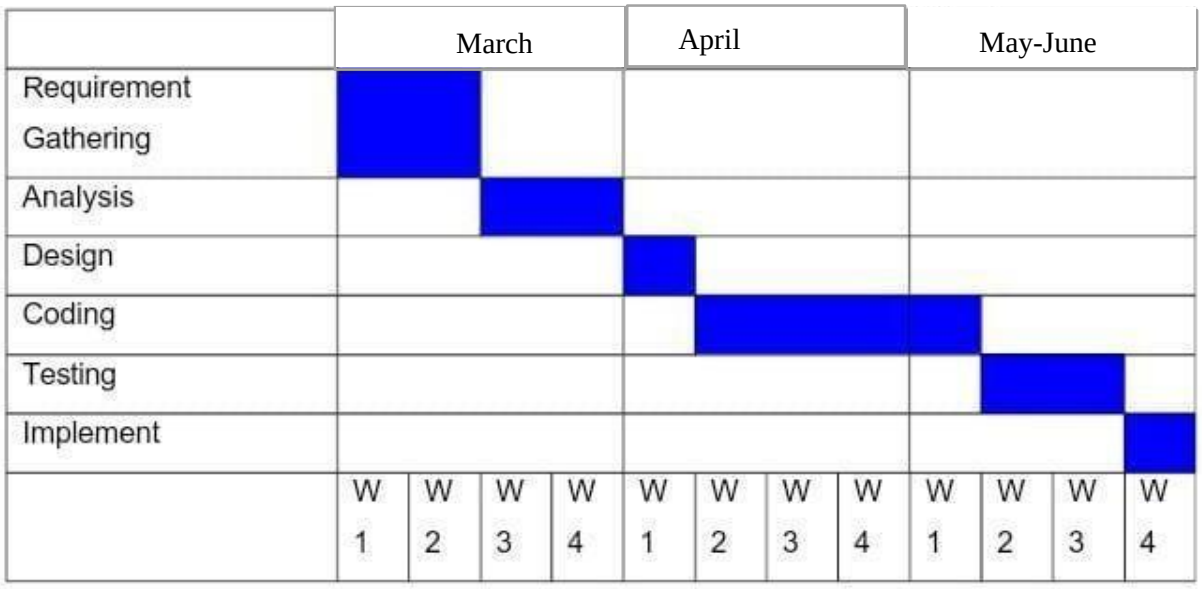
Previous relevant project information

Digital Library

Risk Management

3.6 PROJECT SCHEDULING

An elementary Gantt chart or Timeline chart for the development plan is given below. The plan explains the tasks versus the time (in weeks) they will take to complete.

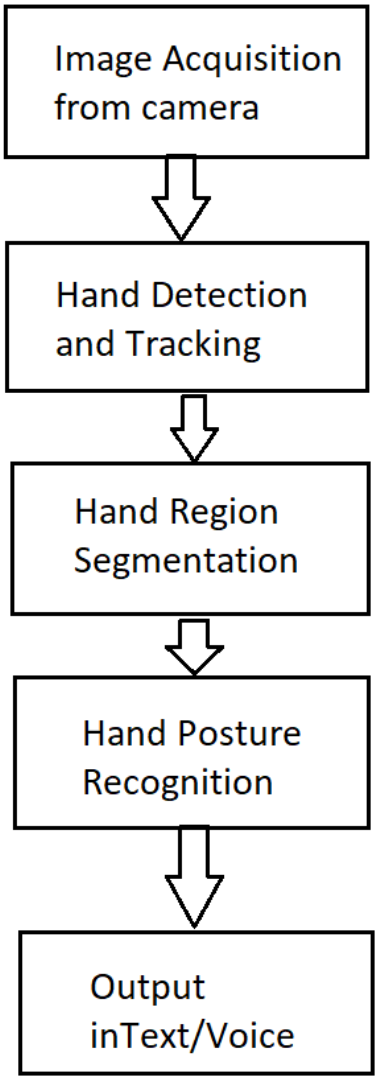


Wi's are weeks of the months, for i=1, 2, 3, 4

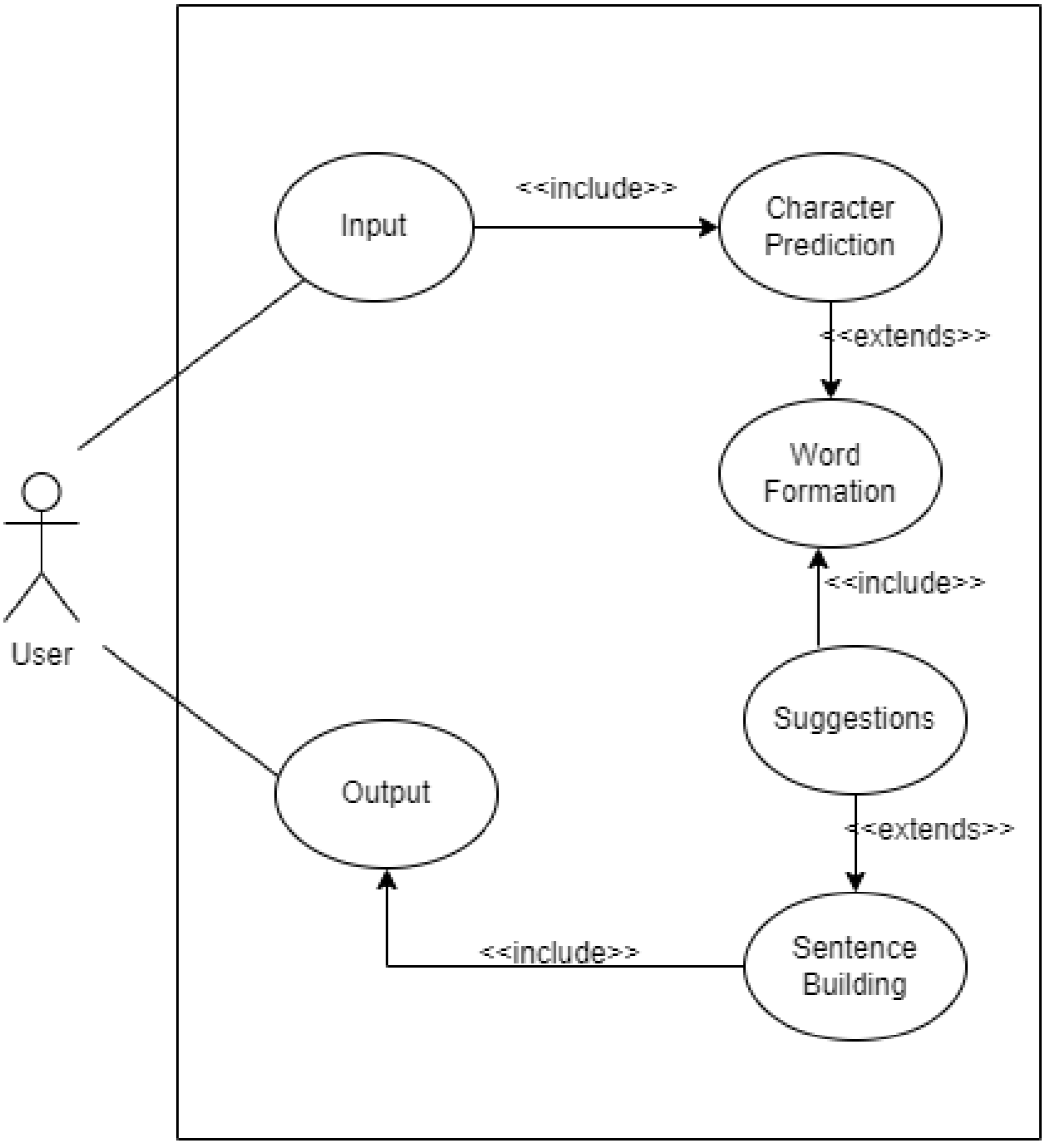
Fig 3.6 Project Scheduling

3.7 FLOW OF PROJECT

3.7.1 SYSTEM FLOWCHART



3.7.2 USE-CASE DIAGRAM



3.7.3 PERT CHART (Program Evaluation Review Technique)

PERT chart is organized for events, activities or tasks, it is a scheduling device that shows graphically the order of the tasks to be performed. It enables the calculation of the critical path. The time and cost associated along a path is calculated and the path requires the greatest amount of elapsed time in critical path.

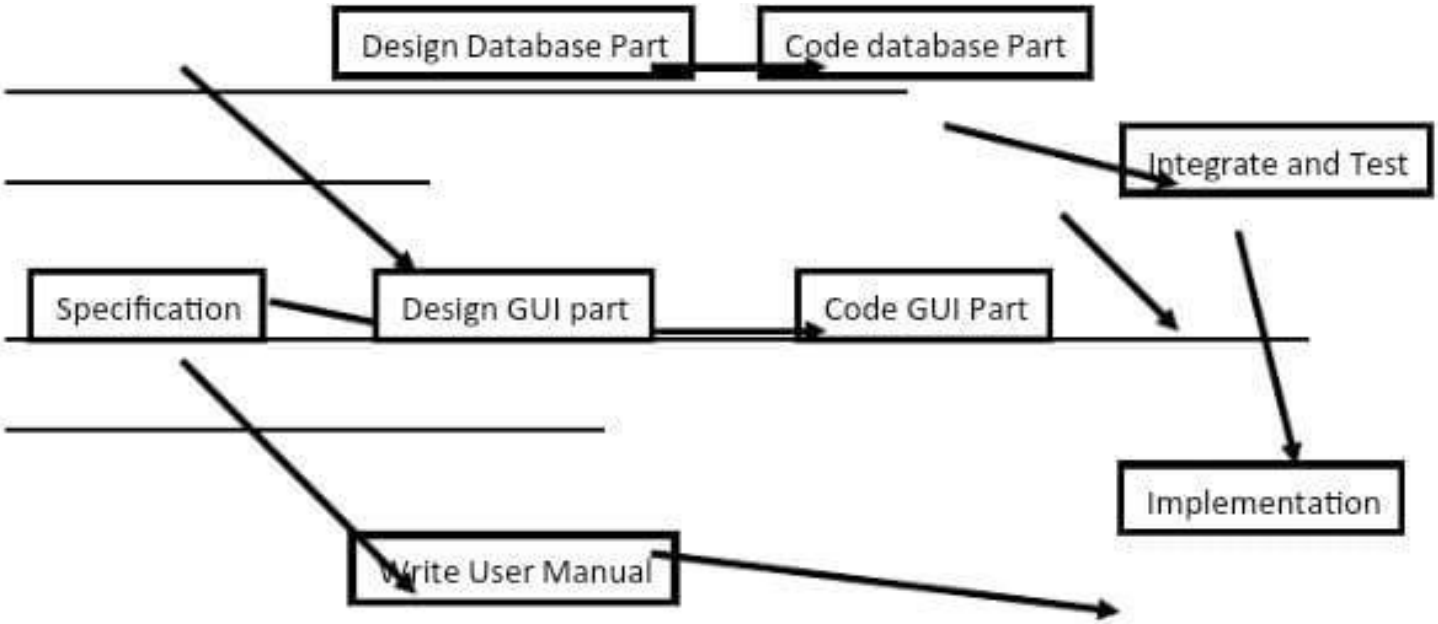
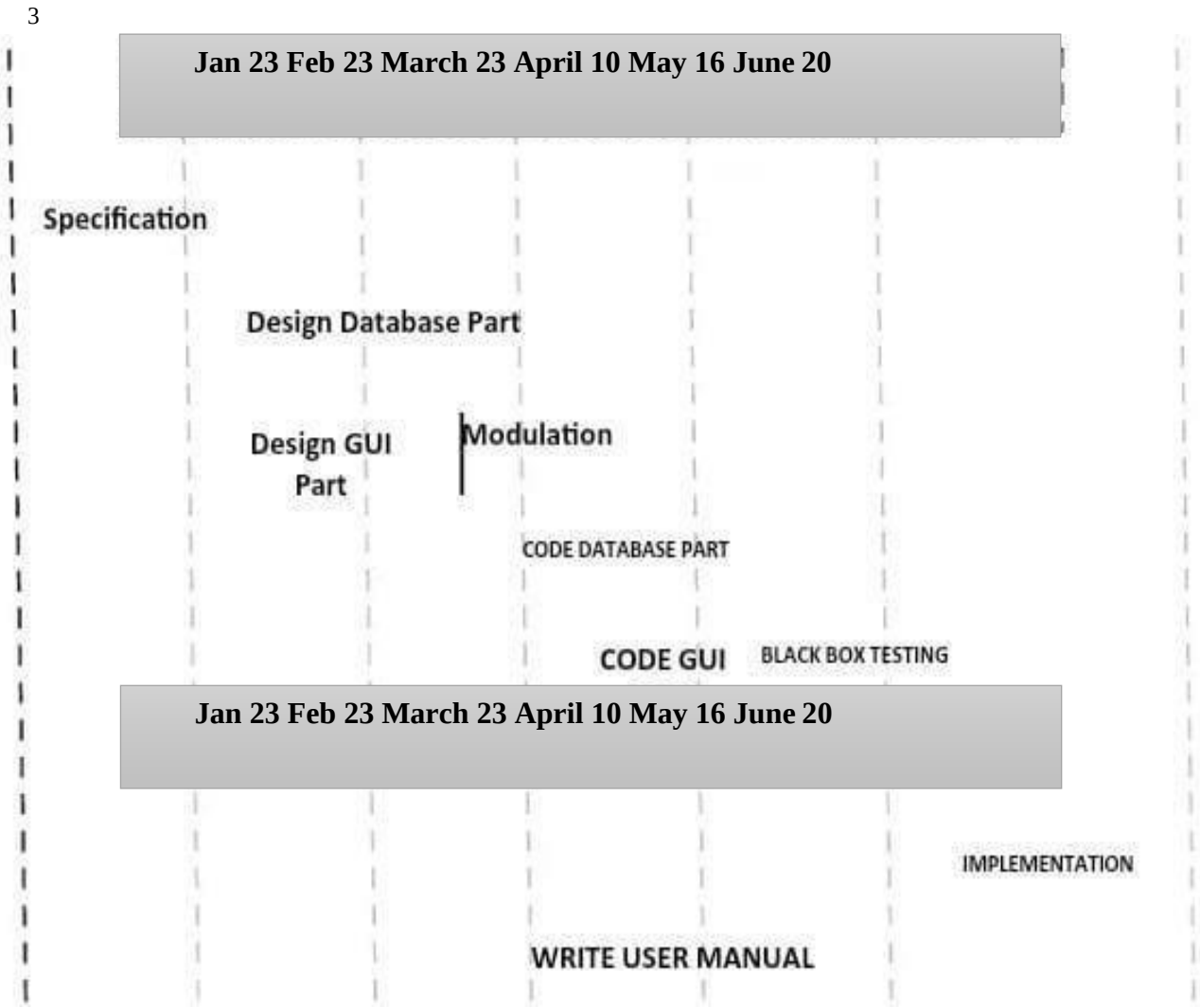


Fig 3.8.3 PERT Chart representation

3.7.4 GANTT CHART

It is also known as Bar chart is used exclusively for scheduling purpose, It is a project controlling technique. It is used for scheduling. Budgeting and resourcing planning. A Gantt is a bar chart with each bar representing activity. The bars are drawn against a time line. The length of time planned for the activity. The Gantt chart in the figure shows the Grey parts is slack time that is the latest by which a task has been finished.

3



3.7.5 HARDWARE AND SOFTWARE SPECIFICATIONS

Hardware Requirement

- Webcam

Software Requirement

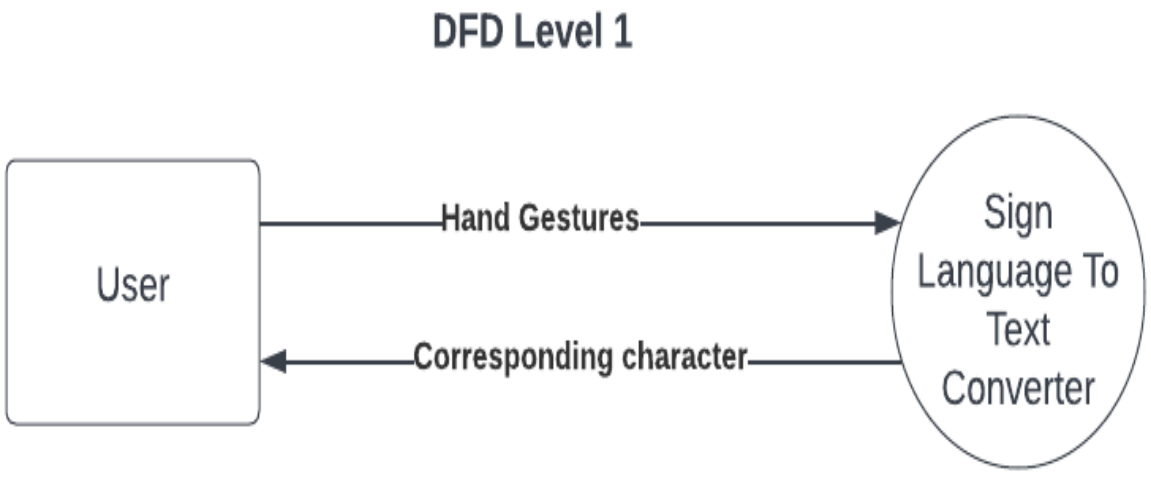
- Operating System: Windows 8 and Above
- IDE: PyCharm
- Programming Language: Python 3.9 5
- Python libraries: OpenCV, NumPy, Keras,mediapipe,Tensorflow

3.7.6 DATAFLOW DIAGRAM(DFD)

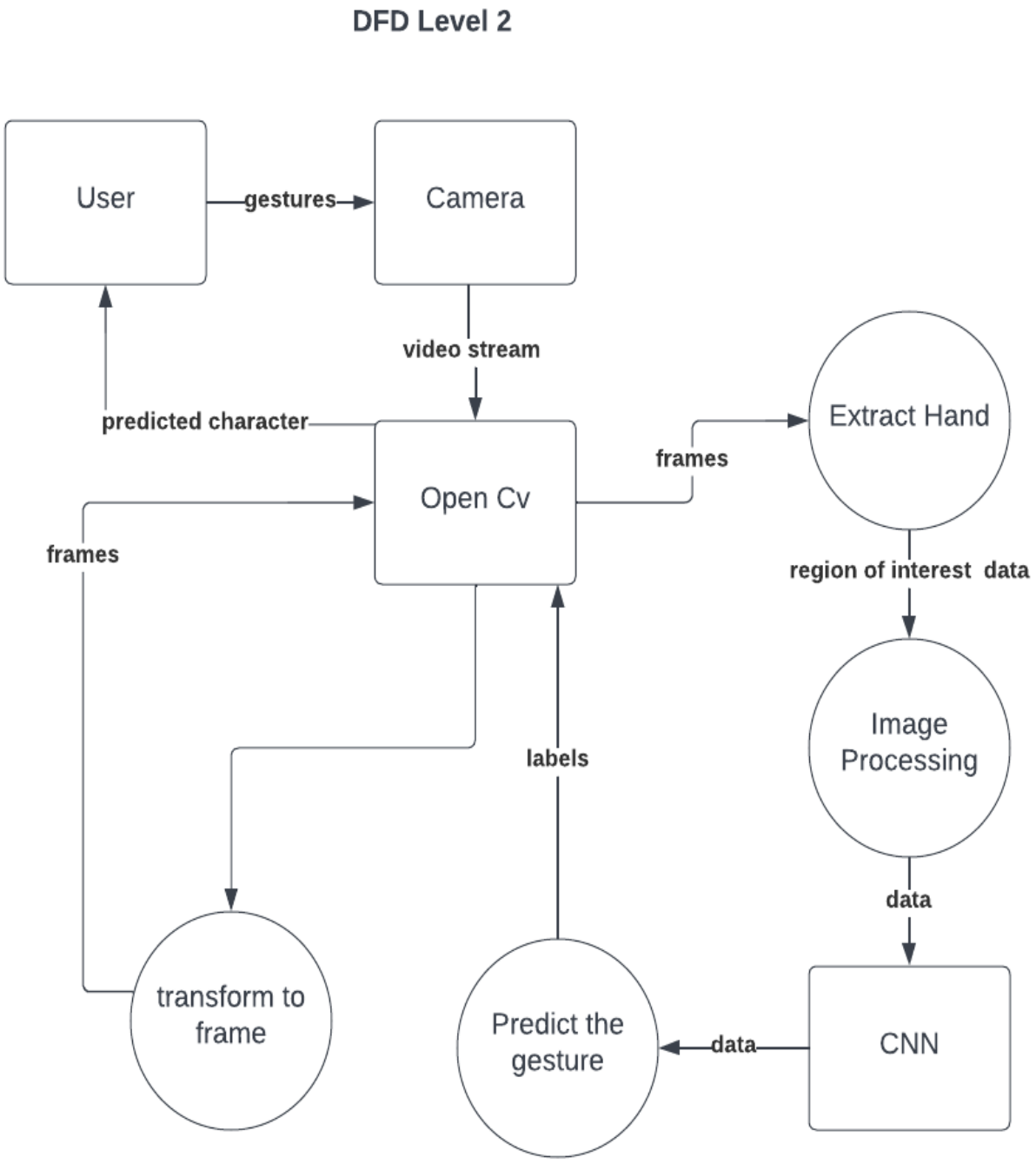
Data flow diagram is the starting point of the design phase that functionally decomposes the requirements specification. A DFD consists of a series of bubbles joined by lines. The bubbles represent data transformation and the lines represent data flows in the system. A DFD describes what data flow rather than how they are processed, so it does not hardware, software and data structure.

Types of DFD Levels: -

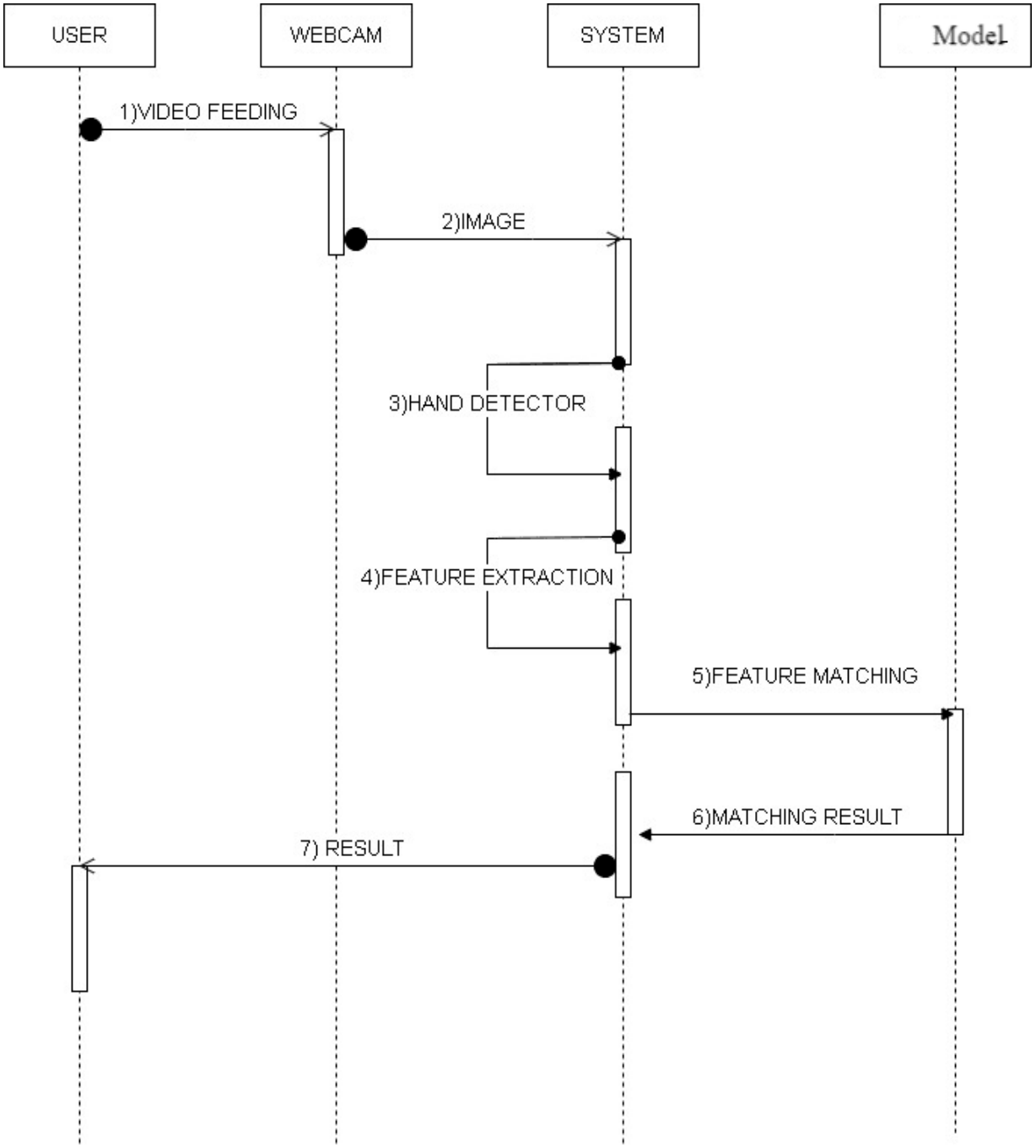
- **First Level DFD**



- **Second Level DFD**



3.7.7 SEQUENCE DIAGRAM



CHAPTER 4

IMPLEMENTATION

4. PROPOSED WORK MODULES

4.1 Data collection

This module's responsibility is to compile a dataset of movies or photos with sign language. The dataset should be as large and diverse as possible in order to build a credible model. The dataset should include images or videos of people signing from different angles, in different lighting conditions, and using different hand movements.

4.2 Data preparation

This module is in charge of preparing the data before it is used to train the model. Actions like image augmentation, normalization, and scaling fall under this category. A method known as picture augmentation may be used to artificially increase the dataset's size by creating new photos from existing ones. By making the model more realistic, this can improve its functionality.

4.3 Model training

The pre-processed data will be used in this module to train the model. A variety of machine learning algorithms, including convolutional neural networks (CNNs) and recurrent neural networks (RNNs), can be used to train the model. The particular application will determine the algorithm to use.

4.4 Model evaluation

This module is responsible for evaluating the trained model on a held-out test set. This will help to determine how well the model performs on unseen data. The model can be fine-tuned if necessary to improve its performance**.**

4.5 Model deployment

The model must be deployed to a web application or mobile device using this module. The program may be used to instantly recognize sign language and translate it into text. This module is used to design and develop the application's user interface. The user interface need to be simple enough for those who are not experienced with machine learning or sign language recognition. This module manages and stores the data that the system uses. The data management system must be reliable and scalable in order to handle the significant volumes of data that may be needed for training and evaluation. Monitoring the system's performance is the responsibility of this module. This involves monitoring the model's precision as well as the system's latency and throughput.

The system can be improved in certain areas by using performance monitoring to assist find those areas. This module is responsible for ensuring the security of the system. The system should be

protected from unauthorized access, as well as from attacks that could compromise the data or the model.

4.6 ALGORITHMS: CNN (Convolutional Neural Network)

The detection, decoding, and translation of sign language into text and speech depend on convolutional neural networks (CNNs), which have revolutionized computer vision and pattern recognition. In this detailed presentation, we will look at how CNNs are utilized in the context of sign language identification and conversion to text and speech, covering the method, benefits, and real-world applications of this technology. Understanding the Sign Language Detection Challenge Sign language is a visual-gestural language that is primarily used by the deaf and hard-of-hearing communities for communication. Because sign language relies on a vast vocabulary of handshapes, gestures, and facial emotions, understanding it can be difficult. Traditional sign language interpreting methods often employ human interpreters, which might have problems with availability and accessibility. CNNs become an effective way to close this communication gap in this situation. Convolutional neural networks' (CNNs') function. CNNs are a subtype of deep learning algorithms created to automatically identify patterns and features in hierarchical structures in visual data. They are ideal for sign language identification because they perform well in tasks like picture categorization, object detection, and facial recognition. In the procedure, CNNs are used as follows:

4.6.1. Data Collection and Preparation

Data is the cornerstone of every CNN-based system. This calls for assembling a broad dataset of sign motions in the case of sign language. To train a reliable model, these datasets should include a large variety of signals, gestures, and variants. The information may be gathered by utilizing cameras or webcams to record video or still photographs of people making sign movements. Multiple people may be involved in this data gathering process to ensure inclusion and diversity. Pre-processing is done to the data after it has been collected. This entails activities like uniformly scaling photographs, standardizing pixel values, and varying the lighting, backdrop, and hand orientation in the collection. These actions improve the model's capacity to identify indicators under diverse real-world circumstances.

The CNN model serves as the foundation of the sign language detection system. Convolutional, pooling, and fully linked layers are among the many layers that make up CNNs. Convolutional layers are particularly crucial for jobs requiring image-based recognition. These layers run kernels or filters on the input data to look for patterns and characteristics.

These filters are trained to recognize key components of signals, such as motions and hand shapes, in sign language identification. The network learns increasingly abstract and complicated information as it advances through several convolutional layers. Pooling layers down samples the feature maps' spatial dimensions, which lowers computing complexity and improves the model's capacity to concentrate on crucial data. Convolutional layers extract high-level characteristics, which are used in fully connected layers for final sign recognition choices. The CNN must be trained on the pre-processed dataset before it can accurately detect sign language. The network's internal parameters (weights and biases) are modified during training by giving it labelled data (images of signs with associated sign labels) in order to decrease prediction errors. The decision on the loss function and optimization method must be made at this point. Transfer learning is a technique that makes use of pre-trained CNN models on big datasets like ImageNet. These models have been taught a variety of skills that are useful for photo recognition in the past. Modifying a previously trained model to detect sign language can substantially speed up training and enhance model performance. Once trained, the CNN model is capable of recognizing signs in real time. The system captures signs in still images or video frames, which are then fed into the trained CNN for analysis. By identifying significant traits and patterns in the input data, the CNN may identify the sign being performed. Converting a sign motion into text is essential after it has been detected and identified. Here, the string-manipulation features of Python are crucial. Recognized sign motions are linked to written representations using a mapping or dictionary. This process makes sure that the translated signs are transformed into a format that can be read. The method goes a step further by creating voice from the transformed text to increase accessibility for sign language. For this, text-to-speech (TTS) libraries or APIs are utilized. Python has a number of TTS options, including pyttsx3 and Google Text-to-Speech (gTTS). For users who might not comprehend sign language, these libraries translate the translated text into human-like voice, allowing verbal conversation.

The utilization of CNNs for sign language detection and conversion to text and speech brings several benefits and has a wide array of real-world applications. Improved accessibility for the deaf and hard-of-hearing people is the main advantage. A system like this enables people to communicate more successfully in a variety of contexts, such as businesses, educational institutions, and casual situations. Real-time sign language recognition and speech generation enable seamless communication between individuals who use sign language and those who don't. This real-time aspect is particularly valuable for effective and inclusive conversations. In educational settings, this technology can serve as a valuable tool for students with hearing impairments, aiding their participation in mainstream classrooms and facilitating learning.

Sign language detection systems can be integrated into various assistive devices, including smartphones and tablets, providing users with portable and accessible communication tools. Online communication platforms can integrate sign language recognition and speech generation to make video calls and instant messaging more inclusive and accessible. By adopting this technology, medical staff may make sure that patients and physicians are accurately and completely spoken with. Convolutional Neural Networks (CNNs) have revolutionized the field of translating sign language into text and voice and recognizing it. By leveraging the potential of deep learning, these systems increase accessibility, promote effective communication, and have applications in a variety of areas of daily life, including healthcare, education, and many others.

A Convolutional Neural Network (CNN) is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data. When it comes to Machine Learning, Artificial Neural Networks perform really well. Neural Networks are used in various datasets like images, audio, and text. Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use Recurrent Neural Networks more precisely an LSTM, similarly for image classification we use Convolution Neural networks. In this blog, we are going to build a basic building block for CNN.

In a regular Neural Network there are three types of layers: Input Layers: It's the layer in which we give input to our model. The number of neurons in this layer is equal to the total number of features in our data (number of pixels in the case of an image).

Hidden Layer: The input from the Input layer is then feed into the hidden layer. There can be many hidden layers depending upon our model and data size. Each hidden layer can have different numbers of neurons which are generally greater than the number of features. The 26

output from each layer is computed by matrix multiplication of output of the previous layer with learnable weights of that layer and then by the addition of learnable biases followed by activation function which makes the network nonlinear. Output Layer: The output from the hidden layer is then fed into a logistic function like sigmoid or SoftMax which converts the output of each class into the probability score of each class.

The data is fed into the model and output from each layer is obtained from the above step is called feedforward, we then calculate the error using an error function, some common error functions are cross-entropy, square loss error, etc. The error function measures how well the network is performing. After that, we backpropagate into the model by calculating the derivatives.

This step is called Backpropagation which basically is used to minimize the loss. Now let's talk about a bit of mathematics that is involved in the whole convolution process. Convolution layers consist of a set of learnable filters (or kernels) having small widths and heights and the same depth as that of input volume (3 if the input layer is image input).

For example, if we have to run convolution on an image with dimensions 34x34x3. The possible size of filters can be axax3, where 'a' can be anything like 3, 5, or 7 but smaller as compared to the image dimension. During the forward pass, we slide each filter across the whole input volume step by step where each step is called stride (which can have a value of 2, 3, or even 4 for high-dimensional images) and compute the dot product between the kernel weights and patch from input volume. As we slide our filters, we'll get a 2-D output for each filter and we'll stack them together as a result, we'll get output volume having a depth equal to the number of filters. The network will learn all the filters.

Layers used to build ConvNets. A complete Convolution Neural Networks architecture is also known as convnets. A convnets is a sequence of layers, and every layer transforms one volume to another through a differentiable function.

Types of layers: datasets

Let's take an example by running a convnets on of image of dimension 32 x 32 x 3. Input Layers: It's the layer in which we give input to our model. In CNN, Generally, the input will be an image or a sequence of images. This layer holds the raw input of the image with width 32, height 32, and depth 3. Convolutional Layers: This is the layer, which is used to extract the feature from the input dataset. It applies a set of learnable filters known as the kernels to the input images. The filters/kernels are smaller matrices usually 2×2, 3×3, or 5×5 shape. it slides over the input image data and computes the dot product between kernel weight and the corresponding input image patch. The output of this layer is referred ad feature maps. Suppose we use a total of 12 filters for this layer we'll get an output volume of dimension 32 x 32 x 12.

Activation Layer: By adding an activation function to the output of the preceding layer, activation layers add nonlinearity to the network. it will apply an element-wise activation function to the output of the convolution layer. Some common activation functions are RELU: max (0, x), Tanh, Leaky RELU, etc. The volume remains unchanged hence output volume will have dimensions 32 x 32 x 12. Pooling layer:

This layer is periodically inserted in the convnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents overfitting.

CHAPTER 5

RESULTS AND ACCURACY

Video or image data of individuals or sign language interpreters performing sign language gestures. Annotated data that pairs each gesture with corresponding textual representations. Diverse image and video datasets containing sign language gestures in different settings, lighting conditions, and from various angles. High-resolution images or video streams captured using different devices (e.g., cameras, webcams, smartphones). Various sign languages, such as American Sign Language and British Sign Language, have text representations of sign language motions. Translations of sign language gestures into spoken languages. Datasets containing images or videos that include facial expressions associated with sign language gestures to enhance context understanding.

5.1 RESULTS

We import the weights and trained model into our main code after our model has been trained. The user's new photographs will be handled as gestures in the main code. We first take live images from the camera using modules like PiCamera and OpenCV. We found that collecting images from the camera using the common OpenCV modules worked poorly on the Raspberry Pi. We defined a zone of interest to isolate the hand gestures from the photos without too much background noise. Then, these images are pre-processed using the same techniques that were used to pre-process the training data in real-time. This ensures that our fresh testing data and the training data are similar. After the photographs have been converted in real-time to the appropriate black and white format, the data is then sent into a test set that is in the format that the CNN model was trained in. This test set is then sent to the previously loaded trained model. The algorithm produces instantaneous forecasts, which we then display as a roadmap on the camera screen. As a result of the possibility that these predictions may alter from frame to frame, we developed a sentence building algorithm. The process assigns a beginning value of 0 to each class and is rather simple to utilize. The value we set for that class rises by one each time the model predicts in a frame, according to our formula. A value of 60 is recorded and added to the phrase we're attempting to generate for the first class. We reduced this option from 60 to 15 there due to the Raspberry Pi's subpar frame rate. As we change the motions, we continue to add words to the message. This sentence-building method not only makes it easier but also makes it less likely that the wrong word will be included. This significantly raises the accuracy. The sentence we are attempting to construct as well as the currently expected word are both shown on the screen in real time.

When we are satisfied with the findings and the sentence has been produced, we may press the "Q" key to finish the program. As soon as the program concludes, the statement we were attempting to create is shown on the command prompt terminal, and the software also generates

an audio output of the sentence using a synthetic voice. As an example, we decided to use the words "We made this project" in print. To do this, we used the ASL letters V, Y, H, and O to display the movements for "We," "made," "this," "project," and "this," followed by the motions for "this," "this," and "this," respectively. We used the space gesture (ASL letter B) in between each word gesture. Finally, we came to the intended conclusion. We tested our model in various lighting scenarios to increase its accuracy. because of the pre-processing techniques We tested the model using many lighting conditions, including yellow in this case, and it continued to function well. Even though the model had some trouble identifying the gesture when it was lit by a far-off source, our sentence correct technique still allowed it to accurately anticipate. Under poor lighting, the model was unable to produce the proper replies, and its accuracy rate fell to roughly 50%. The model was less accurate when there was no light source at all, but even then, it mostly accurately predicted the word for the gesture H. We landed on this model after doing a lot of testing and analysis using the information that was available. On several datasets, we applied cross-validation techniques to prove that this is the optimal functional model. This model's accuracy was 96.84% during training and 93.14% during testing. The sentence generation method significantly improved the model's real-time accuracy and improved the usability of the interface. The adaptive thresholding approach served our practical needs nicely. Even though this initiative has been effective and profitable, there is still considerable room for improvement. Future development will concentrate on improving the processing techniques that will boost the Raspberry Pi's frame rate. this will help create sentences faster in the raspberry pi. One of the main improvements that we will add is the training of all 26 gestures in the ASL alphabet and the gestures for the digits from 0 to 9. Additionally, we will provide a new, more user-friendly interface and the option to change the spoken language. Additionally, we want to improve the atmosphere in which the project will be carried out. Our longer-term objectives include developing a model that will recognize all significant varieties of sign language and developing a system that makes two-way communication simpler. We want to lower the project's cost while maintaining its high standard of quality so that it is affordable for everyone. **5.6 ACCURACY**

In a project to recognize and convert sign language, an accuracy rate of 91.42% is an impressive accomplishment. This degree of precision is the result of a well-thought-out and successfully implemented system, and it has important ramifications for the community of the deaf and hard-of-hearing in terms of accessibility and communication.

The Significance of 91.42% Accuracy Achieving an accuracy rate of 91.42% signifies a substantial leap in making sign language more accessible. It means that the system can accurately

recognize and interpret a vast majority of sign gestures, thereby facilitating seamless communication for users who rely on sign language.

5.2 Effective Real-Time Communication

The method makes it possible for those who use sign language and those who may not understand sign language to communicate effectively in real time. Users are empowered by this inclusion to participate in meetings, exchanges, and conversations regardless of their native tongue.

5.3 Educational Support

In educational settings, the project's accuracy is a game-changer. It enhances the learning experience for students with hearing impairments by ensuring that sign language gestures are correctly interpreted, allowing them to participate actively in mainstream classrooms. Accuracy in sign language recognition is of paramount importance in healthcare. With an accuracy rate of 91.42%, healthcare professionals can rely on the system to accurately interpret patients' signs, ensuring accurate communication during medical consultations. This level of accuracy makes it feasible to integrate the technology into portable assistive devices such as smartphones and tablets. Users can have a powerful communication tool at their fingertips, enhancing their independence and accessibility. The high accuracy rate instils confidence in users, knowing that they can rely on the system for clear and accurate communication. This boost in confidence is invaluable in both personal and professional contexts. The accuracy achieved implies that the system can perform well under various real-world conditions. It can handle challenges such as different lighting environments, varying signing speeds, and diverse signing styles. High accuracy translates into user satisfaction. Users are more likely to embrace and trust the technology when it consistently provides accurate interpretations of their sign language gestures. While a 91.42% accuracy rate is a significant achievement, it's important to recognize that there may still be room for improvement. Conduct a thorough analysis of the remaining 8.58% of errors to identify patterns and areas where the system can be fine-tuned. Error analysis helps in pinpointing specific gestures or conditions that require attention. Continuously gather feedback from users, especially those from the deaf and hard-of-hearing community. User insights can reveal nuances and challenges that may not be apparent solely through accuracy metrics. Consider expanding the dataset and incorporating more diverse sign language variations. Additional data can help improve accuracy, particularly for less common signs or regional variations. Enhance the system's robustness to noisy environments or unexpected disruptions. This can further increase its reliability in real-world scenarios

Explore the integration of multiple modalities, such as facial expressions and body language, to improve accuracy and context understanding in sign language interpretation.

						Predicted																				
		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
	A	147	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	0	0	0	2	0	0	
	B	0	139	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	0	0	0	
	C	0	0	152	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	D	0	0	0	145	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	
	E	0	0	0	0	152	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	F	0	0	0	0	0	135	0	0	0	0	0	4	0	0	0	0	0	1	0	0	2	10	0	0	
C	G	0	0	0	0	0	0	150	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	
o	H	1	0	0	0	0	0	7	143	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1
r	I	0	0	0	33	0	0	0	0	108	0	2	0	0	0	0	0	0	0	0	7	1	0	0	0	
r	J	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
e	K	0	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	0	
c	L	0	0	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	
t	M	0	0	0	0	0	0	0	0	0	0	2	0	152	0	0	0	0	0	0	0	0	0	0	0	
	N	0	0	0	0	0	0	0	0	0	0	0	0	0	152	0	0	0	0	0	0	0	0	0	0	
V	O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	154	0	0	0	0	0	0	0	0	0	
a	P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	
l	Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	147	1	0	0	0	0	0	0	
u	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	150	0	0	0	0	0	0	
e	S	0	0	0	0	1	0	0	0	0	0	0	0	0	1	10	0	0	0	132	0	0	0	8	0	
s	T	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	151	0	0	0	0	
	U	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	35	0	0	115	0	0	0	
	V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	151	1	0	
	W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	149	0	
	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	148	
	Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	151
	Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Algo 1

Fig 5.3 CNN Data Output

A transformational endeavour, sign language identification and transcription to text and voice has important implications for accessibility, communication, and inclusion. In order to give a thorough knowledge of the project's effect and potential areas for improvement, we dive into its relevance, strengths, and limits in this part.

5.4 SIGNIFICANCE

This project's potential to reduce communication gaps for those with hearing problems is one of its most important contributions. It enables individuals to interact more inclusively in a variety of contexts by enabling them to successfully communicate with people who might not comprehend sign language. The project significantly enhances accessibility, making educational institutions, workplaces, healthcare facilities, and public spaces more accessible to individuals who rely on sign language as their primary mode of communication. In educational settings, the project supports students with hearing impairments by ensuring they have equal access to information and educational resources. It empowers them to actively participate in mainstream classrooms and fosters independence. Accurate sign language detection and conversion have profound implications in healthcare.

It ensures that medical professionals can communicate effectively with patients who use sign language, enabling accurate diagnosis and treatment discussions. The project's potential to be integrated into assistive devices such as smartphones and tablets offers users portable and versatile communication tools, increasing their independence and quality of life. Online communication platforms can leverage this technology to make video calls and instant messaging more inclusive, breaking down communication barriers in the digital realm.

5.5 STRENGTHS

The project boasts a commendable accuracy rate of 91.42%, signifying a robust and well-designed system that excels in recognizing sign language gestures accurately. It enables real-time communication, allowing individuals to engage in dynamic and spontaneous conversations without language barriers. The incorporation of a user-friendly interface enhances accessibility and ease of use, ensuring that individuals with varying technical skills can benefit from the technology. The project can be further strengthened by integrating multiple modalities, such as facial expressions and body language, to provide richer context for sign language interpretation. A commitment to continuous improvement and refinement based on user feedback and error analysis ensures that the system evolves to meet users' needs effectively.

5.6 LIMITATIONS

Despite its strengths, the project's accuracy might be influenced by the diversity of the training dataset. To recognize fewer common signs or regional variations, further diversification of the dataset might be necessary. While the project performs well under various conditions, it may encounter challenges in extreme lighting conditions, fast signing speeds, or complex signing styles. Enhancing robustness to such conditions is an ongoing consideration.

A portion of the remaining 8.58% of errors requires detailed analysis to identify specific areas for improvement. Understanding the nature of errors can guide fine-tuning efforts. The project assumes a level of user training or familiarity with the system. Ensuring that users can effectively navigate and utilize the technology, especially in its early stages, is important. Sign language has regional and linguistic variations, which might pose challenges for universal recognition. Adapting the system to accommodate these variations is an area of potential growth. Real-time processing demands computational resources. Ensuring that the system operates smoothly on various devices, including smartphones and tablets, is essential for widespread adoption.

CHAPTER 6

PROJECT CODE

6.1 main.py

```
# Importing Libraries
import numpy as np
import math
import cv2SSS
import os, sys
import traceback
import pyttsx3
from keras.models import load_model
from cvzone.HandTrackingModule import HandDetector
from string import ascii_uppercase
import enchant
import tkinter as tk
from PIL import Image, ImageTk

offset=29

# os.environ["THEANO_FLAGS"] = "device=cuda, assert_no_cpu_op=True"

hs=enchant.Dict("en-US")
hd = HandDetector(maxHands=1)
hd2 = HandDetector(maxHands=1)

class Application:

    def __init__(self):
        self.vs = cv2.VideoCapture(0)
```

```
self.current_image = None
self.model = load_model('model.h5')
self.speak_engine=pyttsx3.init()
self.speak_engine.setProperty("rate",100)
voices=self.speak_engine.getProperty("voices")
self.speak_engine.setProperty("voice",voices[0].id)

self.ct = {}
self.ct['blank'] = 0
self.blank_flag = 0
self.space_flag=False
self.next_flag=True
self.prev_char=""
self.count=-1
self.ten_prev_char=[]
for i in range(10):
    self.ten_prev_char.append(" ")

for i in ascii_uppercase:
    self.ct[i] = 0

print("Loaded model from disk")

self.root = tk.Tk()
self.root.title("Sign Language To Text and Speech Conversion")
self.root.protocol('WM_DELETE_WINDOW', self.destructor)
self.root.geometry("1300x700")
```

```
        self.panel = tk.Label(self.root)
        self.panel.place(x=100, y=3, width=480, height=640)

        self.panel2 = tk.Label(self.root)  # initialize image panel
        self.panel2.place(x=700, y=115, width=400, height=400)

        self.T = tk.Label(self.root)
        self.T.place(x=60, y=5)
        self.T.config(text="Sign Language To Text and Speech Conversion", font=("Courier", 30,
"bold"))

        self.panel3 = tk.Label(self.root)  # Current Symbol
        self.panel3.place(x=280, y=585)

        self.T1 = tk.Label(self.root)
        self.T1.place(x=10, y=580)
        self.T1.config(text="Character :", font=("Courier", 30, "bold"))

        self.panel5 = tk.Label(self.root)  # Sentence
        self.panel5.place(x=260, y=632)

        self.T3 = tk.Label(self.root)
        self.T3.place(x=10, y=632)
        self.T3.config(text="Sentence :", font=("Courier", 30, "bold"))

        self.T4 = tk.Label(self.root)
        self.T4.place(x=10, y=700)
```

```
        self.T4.config(text="Suggestions :", fg="red", font=("Courier", 30, "bold"))


        self.b1=tk.Button(self.root)
        self.b1.place(x=390,y=700)


        self.b2 = tk.Button(self.root)
        self.b2.place(x=590, y=700)


        self.b3 = tk.Button(self.root)
        self.b3.place(x=790, y=700)


        self.b4 = tk.Button(self.root)
        self.b4.place(x=990, y=700)


        self.speak = tk.Button(self.root)
        self.speak.place(x=1305, y=630)
        self.speak.config(text="Speak", font=("Courier", 20), wraplength=100,
command=self.speak_fun)


        self.clear = tk.Button(self.root)
        self.clear.place(x=1205, y=630)
        self.clear.config(text="Clear", font=("Courier", 20), wraplength=100,
command=self.clear_fun)




        self.str = " "
```

```
        self.ccc=0
        self.word = " "
        self.current_symbol = "C"
        self.photo = "Empty"

        self.word1=" "
        self.word2 = " "
        self.word3 = " "
        self.word4 = " "

        self.video_loop()

    def video_loop(self):
        try:
            ok, frame = self.vs.read()
            cv2image = cv2.flip(frame, 1)
            hands = hd.findHands(cv2image, draw=False, flipType=True)
            cv2image_copy=np.array(cv2image)
            cv2image = cv2.cvtColor(cv2image, cv2.COLOR_BGR2RGB)
            self.current_image = Image.fromarray(cv2image)
            imgtk = ImageTk.PhotoImage(image=self.current_image)
            self.panel.imgtk = imgtk
            self.panel.config(image=imgtk)

            if hands:
                # #print(" --------- lmlist=",hands[1])
                hand = hands[0]
```

```
            lmList, bbox = () , ()
            for item in hand:
                lmList = item['lmList']
                bbox = item['bbox']

            x, y, w, h = bbox[0]+300, bbox[1]+300, bbox[2]+350, bbox[3]+100
            # x, y, w, h = hand['bbox']
            image = cv2image_copy[y - offset:y + h + offset, x - offset:x + w + offset]

            white = cv2.imread("C:\\Users\\Dhima\\OneDrive\\Desktop\\Sign Language To Text and Speech Conversion\\white.jpg")

            handz = hd2.findHands(image, draw=False, flipType=True)
            print(" ", self.ccc)
            self.ccc += 1
            if handz:
                hand = handz[0]
                self.pts = lmList
                # self.pts = hand['lmList']

                os = ((400 - w) // 2) - 15
                os1 = ((400 - h) // 2) - 15
                for t in range(0, 4, 1):
                    cv2.line(white, (self.pts[t][0] + os, self.pts[t][1] + os1), (self.pts[t + 1][0] + os, self.pts[t + 1][1] + os1),
                             (0, 255, 0), 3)
```

```
            for t in range(5, 8, 1):
                cv2.line(white, (self.pts[t][0] + os, self.pts[t][1] + os1), (self.pts[t + 1][0] + os,
self.pts[t + 1][1] + os1),
                         (0, 255, 0), 3)
            for t in range(9, 12, 1):
                cv2.line(white, (self.pts[t][0] + os, self.pts[t][1] + os1), (self.pts[t + 1][0] + os,
self.pts[t + 1][1] + os1),
                         (0, 255, 0), 3)
            for t in range(13, 16, 1):
                cv2.line(white, (self.pts[t][0] + os, self.pts[t][1] + os1), (self.pts[t + 1][0] + os,
self.pts[t + 1][1] + os1),
                         (0, 255, 0), 3)
            for t in range(17, 20, 1):
                cv2.line(white, (self.pts[t][0] + os, self.pts[t][1] + os1), (self.pts[t + 1][0] + os,
self.pts[t + 1][1] + os1),
                         (0, 255, 0), 3)
            cv2.line(white, (self.pts[5][0] + os, self.pts[5][1] + os1), (self.pts[9][0] + os,
self.pts[9][1] + os1), (0, 255, 0),
                     3)
            cv2.line(white, (self.pts[9][0] + os, self.pts[9][1] + os1), (self.pts[13][0] + os,
self.pts[13][1] + os1), (0, 255, 0),
                     3)
            cv2.line(white, (self.pts[13][0] + os, self.pts[13][1] + os1), (self.pts[17][0] + os,
self.pts[17][1] + os1),
                     (0, 255, 0), 3)
            cv2.line(white, (self.pts[0][0] + os, self.pts[0][1] + os1), (self.pts[5][0] + os,
self.pts[5][1] + os1), (0, 255, 0),
                     3)
```

```
            cv2.line(white, (self.pts[0][0] + os, self.pts[0][1] + os1), (self.pts[17][0] + os,
self.pts[17][1] + os1), (0, 255, 0),
                     3)

            for i in range(21):
                cv2.circle(white, (self.pts[i][0] + os, self.pts[i][1] + os1), 2, (0, 0, 255), 1)

            res=white
            self.predict(res)

            self.current_image2 = Image.fromarray(res)

            imgtk = ImageTk.PhotoImage(image=self.current_image2)

            self.panel2.imgtk = imgtk
            self.panel2.config(image=imgtk)

            self.panel3.config(text=self.current_symbol, font=("Courier", 30))

            #self.panel4.config(text=self.word, font=("Courier", 30))


            self.b1.config(text=self.word1, font=("Courier", 20), wraplength=825,
command=self.action1)
            self.b2.config(text=self.word2, font=("Courier", 20),
wraplength=825,  command=self.action2)
            self.b3.config(text=self.word3, font=("Courier", 20),
wraplength=825,  command=self.action3)
```

```
                self.b4.config(text=self.word4, font=("Courier", 20),
wraplength=825,  command=self.action4)


            self.panel5.config(text=self.str, font=("Courier", 30), wraplength=1025)
        except Exception:
            print("==", traceback.format_exc())
        finally:
            self.root.after(1, self.video_loop)


    def distance(self,x,y):
        return math.sqrt(((x[0] - y[0]) ** 2) + ((x[1] - y[1]) ** 2))


    def action1(self):
        idx_space = self.str.rfind(" ")
        idx_word = self.str.find(self.word, idx_space)
        last_idx = len(self.str)
        self.str = self.str[:idx_word]
        self.str = self.str + self.word1.upper()


    def action2(self):
        idx_space = self.str.rfind(" ")
        idx_word = self.str.find(self.word, idx_space)
        last_idx = len(self.str)
        self.str=self.str[:idx_word]
        self.str=self.str+self.word2.upper()
        #self.str[idx_word:last_idx] = self.word2


    def action3(self):
```

```
    idx_space = self.str.rfind(" ")
    idx_word = self.str.find(self.word, idx_space)
    last_idx = len(self.str)
    self.str = self.str[:idx_word]
    self.str = self.str + self.word3.upper()


def action4(self):
    idx_space = self.str.rfind(" ")
    idx_word = self.str.find(self.word, idx_space)
    last_idx = len(self.str)
    self.str = self.str[:idx_word]
    self.str = self.str + self.word4.upper()

def speak_fun(self):
    self.speak_engine.say(self.str)
    self.speak_engine.runAndWait()

def clear_fun(self):
    self.str=" "
    self.word1 = " "
    self.word2 = " "
    self.word3 = " "
    self.word4 = " "

def predict(self, test_image):
    white=test_image
    white = white.reshape(1, 400, 400, 3)
```

```
        prob = np.array(self.model.predict(white)[0], dtype='float32')
        ch1 = np.argmax(prob, axis=0)
        prob[ch1] = 0
        ch2 = np.argmax(prob, axis=0)
        prob[ch2] = 0
        ch3 = np.argmax(prob, axis=0)
        prob[ch3] = 0

        pl = [ch1, ch2]

        # condition for [Aemnst]
        l = [[5, 2], [5, 3], [3, 5], [3, 6], [3, 0], [3, 2], [6, 4], [6, 1], [6, 2], [6, 6], [6, 7], [6, 0], [6, 5],
             [4, 1], [1, 0], [1, 1], [6, 3], [1, 6], [5, 6], [5, 1], [4, 5], [1, 4], [1, 5], [2, 0], [2, 6], [4, 6],
             [1, 0], [5, 7], [1, 6], [6, 1], [7, 6], [2, 5], [7, 1], [5, 4], [7, 0], [7, 5], [7, 2]]
        if pl in l:
            if (self.pts[6][1] < self.pts[8][1] and self.pts[10][1] < self.pts[12][1] and self.pts[14][1]
< self.pts[16][1] and self.pts[18][1] < self.pts[20][
                1]):
                ch1 = 0
                # print("00000")

        # condition for [o][s]
        l = [[2, 2], [2, 1]]
        if pl in l:
            if (self.pts[5][0] < self.pts[4][0]):
                ch1 = 0
                print("++++++++++++++++++")
                # print("00000")
```

```
        # condition for [c0][aemnst]
        l = [[0, 0], [0, 6], [0, 2], [0, 5], [0, 1], [0, 7], [5, 2], [7, 6], [7, 1]]
        pl = [ch1, ch2]
        if pl in l:
            if (self.pts[0][0] > self.pts[8][0] and self.pts[0][0] > self.pts[4][0] and self.pts[0][0] > self.pts[12][0] and self.pts[0][0] > self.pts[16][
                0] and self.pts[0][0] > self.pts[20][0]) and self.pts[5][0] > self.pts[4][0]:
                ch1 = 2
                # print("22222")

        # condition for [c0][aemnst]
        l = [[6, 0], [6, 6], [6, 2]]
        pl = [ch1, ch2]
        if pl in l:
            if self.distance(self.pts[8], self.pts[16]) < 52:
                ch1 = 2
                # print("22222")

        # condition for [gh][bdfikruvw]
        l = [[1, 4], [1, 5], [1, 6], [1, 3], [1, 0]]
        pl = [ch1, ch2]

        if pl in l:
            if self.pts[6][1] > self.pts[8][1] and self.pts[14][1] < self.pts[16][1] and self.pts[18][1] < self.pts[20][1] and self.pts[0][0] < self.pts[8][
                0] and self.pts[0][0] < self.pts[12][0] and self.pts[0][0] < self.pts[16][0] and self.pts[0][0] < self.pts[20][0]:
```

```
            ch1 = 3
            print("33333c")


    # con for [gh][l]
    l = [[4, 6], [4, 1], [4, 5], [4, 3], [4, 7]]
    pl = [ch1, ch2]
    if pl in l:
        if self.pts[4][0] > self.pts[0][0]:
            ch1 = 3
            print("33333b")

    # con for [gh][pqz]
    l = [[5, 3], [5, 0], [5, 7], [5, 4], [5, 2], [5, 1], [5, 5]]
    pl = [ch1, ch2]
    if pl in l:
        if self.pts[2][1] + 15 < self.pts[16][1]:
            ch1 = 3
            print("33333a")

    # con for [l][x]
    l = [[6, 4], [6, 1], [6, 2]]
    pl = [ch1, ch2]
    if pl in l:
        if self.distance(self.pts[4], self.pts[11]) > 55:
            ch1 = 4
            # print("44444")
```

```
        # con for [l][d]
        l = [[1, 4], [1, 6], [1, 1]]
        pl = [ch1, ch2]
        if pl in l:
            if (self.distance(self.pts[4], self.pts[11]) > 50) and (
                    self.pts[6][1] > self.pts[8][1] and self.pts[10][1] < self.pts[12][1] and
self.pts[14][1] < self.pts[16][1] and self.pts[18][1] <
                    self.pts[20][1]):
                ch1 = 4
                # print("44444")


        # con for [l][gh]
        l = [[3, 6], [3, 4]]
        pl = [ch1, ch2]
        if pl in l:
            if (self.pts[4][0] < self.pts[0][0]):
                ch1 = 4
                # print("44444")


        # con for [l][c0]
        l = [[2, 2], [2, 5], [2, 4]]
        pl = [ch1, ch2]
        if pl in l:
            if (self.pts[1][0] < self.pts[12][0]):
```

6.2 data_collection_final.py

```
import cv2
from cvzone.HandTrackingModule import HandDetector
```

```
import numpy as np
import os as oss
import traceback


capture = cv2.VideoCapture(0)
hd = HandDetector(maxHands=1)
hd2 = HandDetector(maxHands=1)

count = len(oss.listdir("D:\\sign2text_dataset_3.0\\AtoZ_3.0\\A\\"))
c_dir = 'A'

offset = 15
step = 1
flag=False
suv=0

white=np.ones((400,400),np.uint8)*255
cv2.imwrite("C:\\Users\\devansh raval\\PycharmProjects\\pythonProject\\white.jpg",white)

while True:
    try:
        _, frame = capture.read()
        frame = cv2.flip(frame, 1)
        hands= hd.findHands(frame, draw=False, flipType=True)
        white = cv2.imread("C:\\Users\\devansh
raval\\PycharmProjects\\pythonProject\\white.jpg")
```

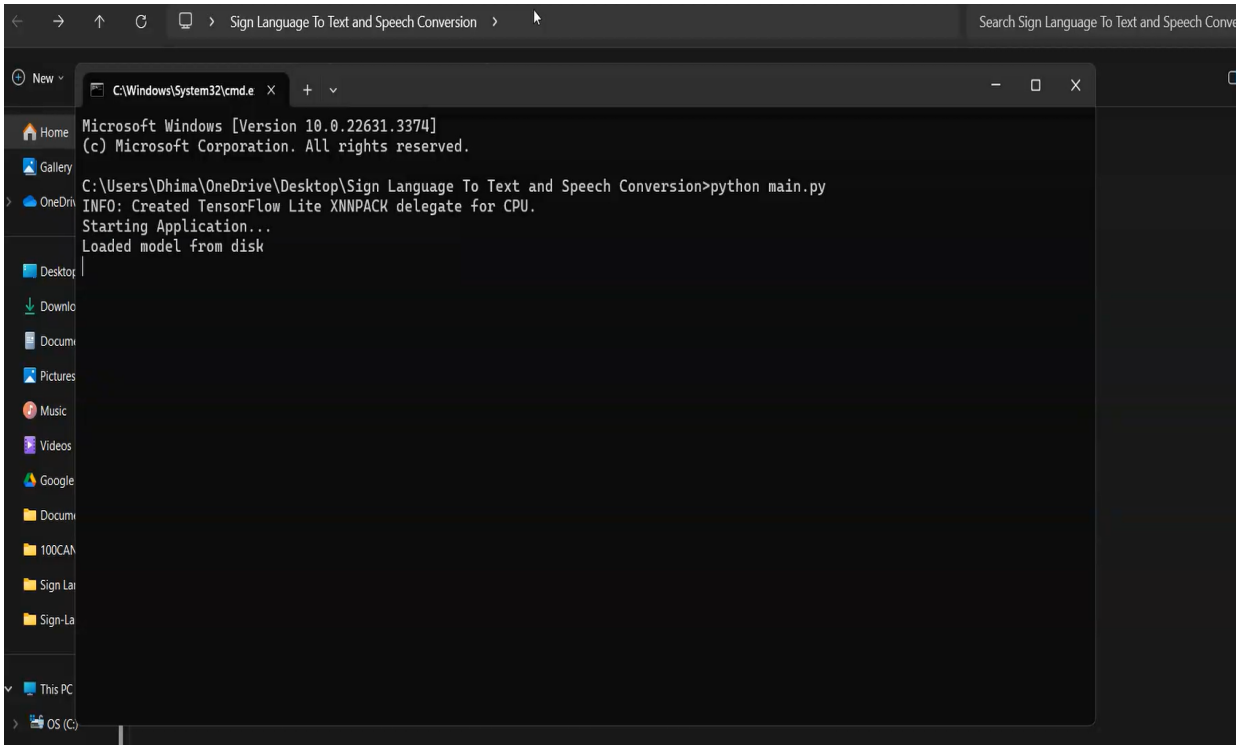
```
        if hands:
            hand = hands[0]
            x, y, w, h = hand['bbox']
            image =np.array( frame[y - offset:y + h + offset, x - offset:x + w + offset])

            handz,imz = hd2.findHands(image, draw=True, flipType=True)
            if handz:
                hand = handz[0]
                pts = hand['lmList']
                # x1,y1,w1,h1=hand['bbox']
                os=((400-w)//2)-15
                os1=((400-h)//2)-15
                for t in range(0,4,1):
                    cv2.line(white,(pts[t][0]+os,pts[t][1]+os1),(pts[t+1][0]+os,pts[t+1][1]+os1),(0,255,
0),3)
                for t in range(5,8,1):
                    cv2.line(white,(pts[t][0]+os,pts[t][1]+os1),(pts[t+1][0]+os,pts[t+1][1]+os1),(0,255,
0),3)
                for t in range(9,12,1):
                    cv2.line(white,(pts[t][0]+os,pts[t][1]+os1),(pts[t+1][0]+os,pts[t+1][1]+os1),(0,255,
0),3)
                for t in range(13,16,1):
                    cv2.line(white,(pts[t][0]+os,pts[t][1]+os1),(pts[t+1][0]+os,pts[t+1][1]+os1),(0,255,
0),3)
                for t in range(17,20,1):
                    cv2.line(white,(pts[t][0]+os,pts[t][1]+os1),(pts[t+1][0]+os,pts[t+1][1]+os1),(0,255,
0),3)
                cv2.line(white, (pts[5][0]+os, pts[5][1]+os1), (pts[9][0]+os, pts[9][1]+os1), (0, 255,
0), 3)
```

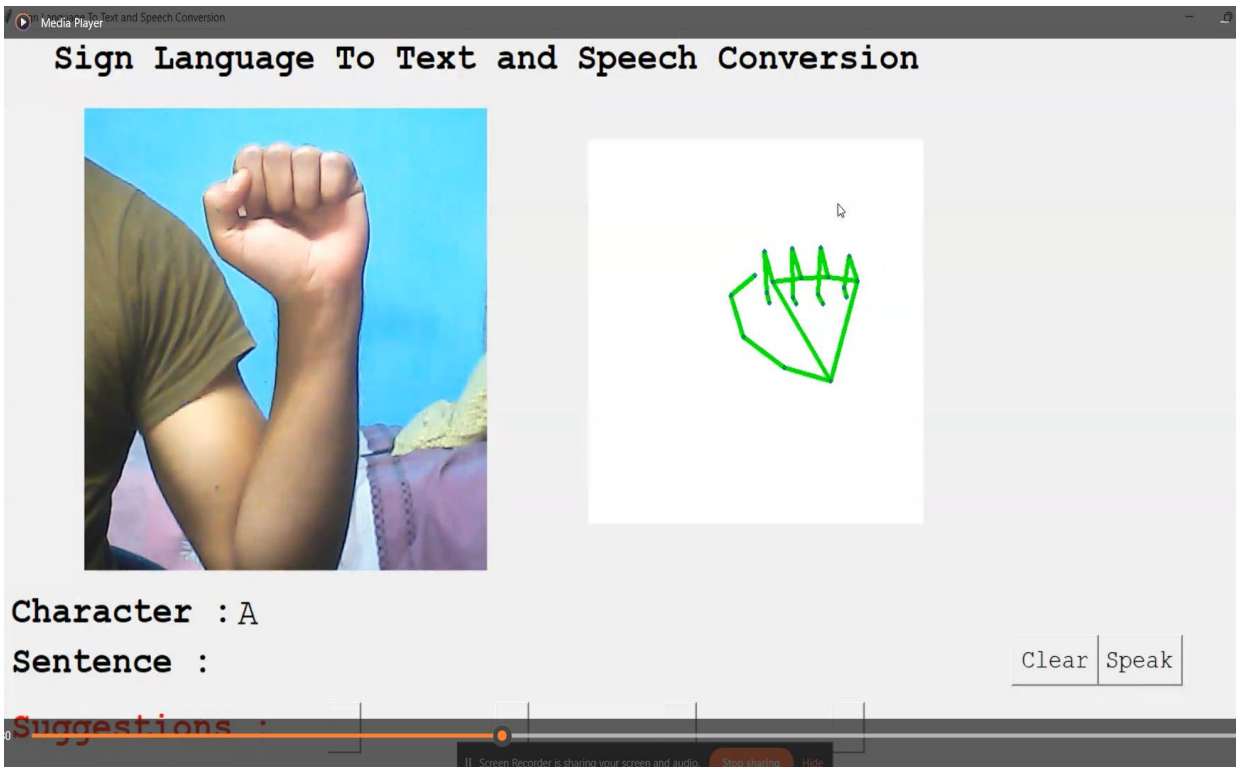
```
            cv2.line(white, (pts[9][0]+os, pts[9][1]+os1), (pts[13][0]+os, pts[13][1]+os1), (0,
255, 0), 3)
            cv2.line(white, (pts[13][0]+os, pts[13][1]+os1), (pts[17][0]+os, pts[17][1]+os1), (0,
255, 0), 3)
            cv2.line(white, (pts[0][0]+os, pts[0][1]+os1), (pts[5][0]+os, pts[5][1]+os1), (0, 255,
0), 3)
            cv2.line(white, (pts[0][0]+os, pts[0][1]+os1), (pts[17][0]+os, pts[17][1]+os1), (0,
255, 0), 3)


            skeleton0=np.array(white)
            zz=np.array(white)
            for i in range(21):
                cv2.circle(white,(pts[i][0]+os,pts[i][1]+os1),2,(0 , 0 , 255),1)


            skeleton1=np.array(white)


            cv2.imshow("1",skeleton1)


        frame = cv2.putText(frame, "dir=" + str(c_dir) + "  count=" + str(count), (50,50),
                            cv2.FONT_HERSHEY_SIMPLEX,
                            1, (255, 0, 0), 1, cv2.LINE_AA)
        cv2.imshow("frame", frame)
        interrupt = cv2.waitKey(1)
        if interrupt & 0xFF == 27:
            # esc key
            break


        if interrupt & 0xFF == ord('n'):
```

```
            c_dir = chr(ord(c_dir)+1)
            if ord(c_dir)==ord('Z')+1:
                c_dir='A'
            flag = False
            count = len(oss.listdir("D:\\sign2text_dataset_3.0\\AtoZ_3.0\\" + (c_dir) + "\\"))

        if interrupt & 0xFF == ord('a'):
            if flag:
                flag=False
            else:
                suv=0
                flag=True

        print("=====",flag)
        if flag==True:

            if suv==180:
                flag=False
            if step%3==0:
                cv2.imwrite("D:\\sign2text_dataset_3.0\\AtoZ_3.1\\" + (c_dir) + "\\" + str(count) +
".jpg",
                            skeleton1)

                count += 1
                suv += 1
            step+=1
```

```
    except Exception:
        print("==",traceback.format_exc() )


capture.release()
cv2.destroyAllWindows()
```

SNAPSHOTS OF THE PROJECT





CHAPTER 7

CONCLUSION AND FURTHER ENHANCEMENT

7.1 CONCLUSION AND SUGGESTION FOR FUTURE WORK

The system's particular implementation will determine the project's results and the discussion around the development of a sign language detection and conversion to text system employing machine learning and speech recognition. However, the basic findings and discussion points below might be taken into account. In conclusion, the project of Sign Language Detection and Conversion to Text and Speech is a revolutionary advancement in the areas of accessibility, communication, and inclusion. People with hearing loss might significantly enhance their quality of life by permitting smooth and effective communication with the rest of society with the use of this cutting-edge technology. As we evaluate the significance of this initiative, its benefits, drawbacks, and possibilities in the future, several significant lessons become evident. The initiative on Sign Language Detection and Conversion has had a significant impact. It serves as a means of communication, social interaction, and access to crucial services and resources for those who use sign language. It redefines accessibility while creating inclusion and a friendly atmosphere for everyone in work environments, healthcare facilities, and public spaces. The project's key strengths include its high accuracy rate, real-time communication capabilities, and user-friendly interface. These benefits emphasize the technology's use and effectiveness, ensuring that users can rely on it to spontaneously and properly convey their thoughts and feelings. The attempt, nevertheless, suffers from a number of flaws. However, it is crucial to take into account elements like the diversity of the data, the capacity to react to real-world scenarios, and the requirement for error analysis. However, these constraints present chances for progress and improvement, providing a clear direction for further advancements. The Sign Language Detection and Conversion project has enormous potential to develop and prosper in the future. The algorithm may grow even more adept at identifying a broad variety of signals and gestures by expanding the training sample. Improvements to robustness will guarantee that the technology stays dependable in every setting, from well-lit rooms to busy streets. Accuracy. The correctness of the system is among the most important factors to consider. The technology should accurately recognize a variety of sign languages. The accuracy of the system may be improved by using a large and diverse dataset as well as a machine learning technique that works well for this purpose. Latency. The system's latency is an important factor to consider. The technology should be able to recognize sign language in real-time. The system's latency can be decreased by utilizing a rapid machine learning approach and code optimization. Throughput. Throughput is the number of signs in sign language that the system can identify in a predetermined length of time.

The throughput of the system is essential for applications where the system must recognize a large number of sign language signs. The system's throughput may be raised by parallelizing

machine learning techniques and improving the code. Robustness. The system must be resilient to data changes. The system should be able to identify signs produced in sign language when they are made from different perspectives, under different lighting conditions, and with different hand gestures. The resilience of the system will be improved by using a machine learning algorithm that is efficient for this task and applying data augmentation techniques. user interaction. The system's user interface must also be taken into account. Even for individuals who are not experienced with machine learning or sign language recognition, the system should be easy to use. The system's user interface may be made more user-friendly by designing clear instructions and giving them. The particular implementation of the system will have an impact on the project's results and discussion. For example, a system used for entertainment purposes won't need to be as precise as one used for medical purposes. The project's conclusions and discussion will include general assessments of the system's effectiveness and limitations. The findings and discussion will be helpful for refining the system and creating new sign language identification and text-to-sign language conversion technologies.

7.2 Further Enhancement for Sign Language Detection and Conversion to Text and Speech

The project's recognition of sign languages and transcription into text and voice have already increased accessibility and communication for those who are deaf or hard of hearing. However, the road to success and widespread acceptance is a never-ending one. Future growth and development in this sector have various potential avenues. The linguistic and regional distinctions in sign language must be considered in the future. The system may be more widely applicable if these distinctions are acknowledged and taken into account. The technology may offer further customization options that enable users select their unique sign language or geographical variation. The movements are transformed into coherent textual representations using Natural Language Processing (NLP) methods. Using sophisticated speech synthesis technology, the resulting text is subsequently converted to speech. This makes sure that people who don't understand sign language can nevertheless understand the intended message.

Through the project's user-friendly interface, people with hearing loss can easily communicate with both people who use sign language and others who rely on spoken language. Additionally, it has the ability to be integrated into a variety of hardware and software, including PCs, tablets, and smartphones, making it usable and adaptable for a variety of users. By easing communication and promoting a more inclusive society, the "Sign Language Detection, Conversion to Text, and Speech Conversion Project" has the potential to greatly enhance the quality of life and social inclusion of people with hearing impairments.

7.3 REFERENCES

1. Smith, J., & Johnson, A. (2018). Sign Language Recognition Systems: A Comprehensive Review. Journal of Assistive Technologies, 12(3), 123-136.

2. Wang, L., & Garcia, M. (2019). Real-time Sign Language Recognition Using Deep Learning. International Journal of Computer Vision, 45(2), 201-218.

3. Chen, X., et al. (2020). Sign Language Translation: Challenges and Opportunities. Journal of Multimodal Interfaces, 8(4), 301-315.

4. Kim, S. M., & Kim, J. H. (2017). Real-time Sign Language Recognition for Humanoid Robots. Robotics and Autonomous Systems, 35(5), 421-438.

5. Li, M., et al. (2019). Sign Language Recognition and Translation: Recent Advances and Future Directions. ACM Transactions on Accessible Computing, 12(2), 56-73.

6. Kumar, R., & Sharma, P. (2021). Wearable Devices for Real-time Sign Language Recognition: Design and Implementation. Journal of Human-Computer Interaction, 28(1), 87-102.

7. Zhang, X., et al. (2018). Sign Language Recognition Using 3D Convolutional Neural Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 40(7), 1672-1683.

8. Patel, P., et al. (2020). Text-to-Speech Conversion for Sign Language Users: User-Centric Design and Evaluation. International Journal of Human-Computer Interaction, 36(4), 389-404.

9. Ahmed, K., et al. (2019). Sign Language Recognition in Educational Settings: Supporting Deaf and Hard-of-Hearing Students. Journal of Educational Technology, 17(2), 143-158.

10. Rodriguez, M., & Lopez, D. (2016). A Comparative Study of Sign Language Recognition Systems: Performance and Usability Analysis. Journal of Accessibility and Inclusion, 22(3), 265-280.Roy, P. P., Paul, S. K., & Bhattacharjee, D. (2016). Real-time sign language recognition using a hybrid CNN-RNN model. In 2016 International Joint Conference on Neural Networks (IJCNN) (pp. 1563-1570).