CoSign: Sign Language Translator Extension

Submitted in partial fulfillment of the requirements of the degree of

BACHELOR OF COMPUTER ENGINEERING

by

Aastha Patil - 20102083

Anwesha Pani - 20102063

Vaishnavi Kothari - 20102138

Chetan Kumbhar - 20102101

Guide:

Prof. Ashhvini Gaikwad

Department of Computer Engineering

A. P. SHAH INSTITUTE OF TECHNOLOGY, THANE

(2022-2023)

A. P. SHAH INSTITUTE OF TECHNOLOGY, THANE

CERTIFICATE

This is to certify that the Mini Project 2B entitled "**CoSign: Sign Language Translator Extension**" is a bonafide work of **" Aastha Patil (20102083), Anwesha Pani (20102063) ,Vaishnavi Kothari (20102138), Chetan Kumbhar (20102101) "** submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of **Bachelor of Engineering** in **Computer Engineering.**

Guide	Project Coordinator	Head of Department
Prof. Ashvini A. Gaikwad	Prof. D.S. Khachane	Prof. S.H. Malave

A. P. SHAH INSTITUTE OF TECHNOLOGY, THANE

Project Report Approval for Mini Project-2B

This project report entitled **"CoSign: Sign Language Translator Extension"** by ***Aastha Patil , Anwesha Pani, Vaishnavi Kothari, Chetan Kumbhar*** is approved for the partial fulfillment of the degree of ***Bachelor of Engineering*** in ***Computer Engineering***, ***2022-23***.

Examiner Name	Signature
1._____	_____
2._____	_____

Date:

Place:

Declaration

We declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Aastha Patil – 20102083

Anwesha Pani – 20102063

Vaishnavi Kothari – 20102138

Chetan Kumbhar – 20102101

Date:

Abstract

CoSign: Sign Language Translator Extension is a software solution designed to detect American Sign Language (ASL) signs during a Google Meet. This application's primary goal is to establish seamless communication between hearing- and speech-impaired individuals and able individuals. Currently, few browser extensions provide this feature during an ongoing Google Meet. The extension utilizes an LSTM-based deep learning model which makes use of OpenCV and Mediapipe holistic to extract key points from our hands and faces and train our model using these key points. We use web sockets to establish a bi-directional connection between client and server to send the frames of the action to the server, making predictions and in turn receive the predicted action from the server. The signer will keep his video on and frames will be extracted from the ongoing Google Meet. The non-signer will enable the extension to understand the sign and will share his screen in the extension to interpret the sign.

Keywords: ASL, Google Meet Extension, LSTM model, OpenCV, Mediapipe.

CONTENTS

LIST OF FIGURES

Chapter 1
Introduction

Sign language is a visual language used to communicate with individuals who are deaf, hard of hearing, or mute. It is a complete language, with its grammar, syntax, and vocabulary. Each sign in sign language is unique due to variations in hand form, motion profile, and positioning of the hand, face, and other body components. There are numerous distinct sign languages, each of which has a distinct vocabulary and syntax. For example, British Sign Language (BSL) is used in the UK while American Sign Language (ASL) is the most common sign language used in the United States and Canada. In addition, there are regional variations of sign language, such as Mexican Sign Language or Japanese Sign Language.

It has often been difficult for deaf and mute people to communicate with abled people. As we know people with hearing and speaking impairment have to rely on sign language for daily communication. According to the World Federation of the Deaf, there are more than 70 million deaf people worldwide. More than 80% of them live in developing countries. Collectively, they use more than 300 different sign languages. Not only the deaf and mute people but also the abled who interact with them need to have a basic understanding of sign languages. Not only do the signers and non-signers need to understand sign language but they also have to deal with the numerous sign languages used worldwide. This in turn creates confusion during communication for everyone. It is a tedious and difficult task to remember all the signs in any particular sign language. Also, people have to constantly rely on a sign language human interpreter who may not always be available.

For in-person meetings, hosts and employers are required by the Americans with Disabilities Act (ADA) to offer communication accessibility services, such as on-site sign language interpreters or Computer Assisted Realtime Translation (CART). Each user who is hard of hearing or deaf must have their individual needs met by the communication method used. Public hosts must make sure that their services are totally accessible in order to uphold their legal obligations under accessibility regulations. Hence, during offline conferences and meetings it is easier for the disabled people to easily understand and communicate with the abled people. Even news channels, concerts, public announcements all have sign language translators and interpreters.

Many communication-related technologies have come up with the concept of using Machine Learning to bridge the communication gap between the hard of hearing and speech-impaired people and hearing and speaking individuals. Many of them have also been successful in doing so. There are sign language supporting, detecting, and translating applications and even extensions that allow users to overlay videos of sign language on platforms like Disney+, Netflix, and even browsers. But it is observed that there are very few extensions that allow sign language interpretation and translation on popular online meeting platforms like Google Meet, Zoom, etc.

With the advancing technology, the use of virtual meeting apps is increasing. People now often conduct meetings with other people from any part of the world through video conferencing platforms. In these web conferences, some people cannot hear or speak. Here is where the issue arises. For them, it becomes difficult to communicate virtually. As observed, virtual meeting platforms like Google Meet have fewer provisions or support for interpreting sign language. In accordance with the Twenty-First Century Communications and Video Accessibility Act of 2010 (CVAA) [1], interoperable video conferencing services must be accessible to people with disabilities, however both the meaning of "interoperable" and what it means to be "accessible" are up for debate.

Hence, we have come up with a solution that will work as an extension of the Google Meet platform and provide assistance to the hearing and speech impaired as well as non-impaired individuals via online video conference, thus establishing a way of communication between both types efficiently.

Chapter 2

Literature Survey

1. **American Sign Language Recognition using Deep Learning and Computer Vision:**

   This article describes how artificial intelligence and machine learning can be used to bridge the communication gap between people with speech disorders and the general public. Such people with disabilities use sign language for making social connections. To ease the interaction, they created a vision-based application that offers sign language translation to text. They used machine learning concepts such as CNN, and RNN for building models.

2. **Large Scale Sign Language Interpretation:**

   This paper focuses on the following fact that Sign language is the primary means of communication among deaf people, but the majority of hearing people do not know sign language. They present the world's largest sign language dataset to date. This is a collection of 50,000 video snippets from a pool of 10,000 unique utterances signed by 50 signers. They further propose several sequence-to-sequence deep learning approaches for automatically translating from Chinese sign language to English and Mandarin text. Although the model may overfit the training set, generalizing to unexpected utterances using real-world data is still a challenge.

3. **Recent Advances of Deep Learning for Sign Language Recognition**

   This paper provides an overview of deep learning-based techniques for sign language recognition. They describe different approaches developed for site-to-view modality detection provided by depth sensors, feature extraction, and classification. In addition, we summarize currently available sign language datasets, such as finger gestures and vocabulary, that can be used as assessment tools for sign language learners.

Sr. No.	Title	Authors	Description
1)	American Sign Language Recognition using Deep Learning and Computer Vision - January 2019	1. Ying Xie 2. Kshitij Bantupalli	This article describes how artificial intelligence and machine learning can be used to bridge the communication gap between people with speech disorders and the general public. Such people with disabilities use sign language for making social connections. To ease the interaction, they created a vision-based application that offers sign language translation to text. They used machine learning concepts such as CNN, and RNN for building models.
2)	Large Scale Sign Language Interpretation - July 2019	1. Tiantian Yuan 2. Shagan Sah 3. Tejaswini Ananthanarayana 4. Chi Zhang 5. Aneesh Bhat 6. Sahaj Gandhi 7. Raymond Ptucha	This paper focuses on the following fact that Sign language is the primary means of communication among deaf people, but the majority of hearing people do not know sign language. They present the world's largest sign language dataset to date. This is a collection of 50,000 video snippets from a pool of 10,000 unique utterances signed by 50 signers. They further propose several sequence-to-sequence deep learning approaches for automatically translating from Chinese sign language to English and Mandarin text. Although the model may overfit the training set, generalizing to unexpected utterances using real-world data is still a challenge.
3)	Recent Advances of Deep Learning for Sign Language Recognition - December 2019	1. Lihong Zheng 2. Bin Liang 3. Ailian Jiang	This paper provides an overview of deep learning-based techniques for sign language recognition. They describe different approaches developed for site-to-view modality detection provided by depth sensors, feature extraction, and classification. In addition, we summarize currently available sign language datasets, such as finger gestures and vocabulary, that can be used as assessment tools for sign language learners.

Chapter 3

Problem Statement, Objective & Scope

3.1 Problem Statement: -

To design and implement a Google Meet extension for American Sign Language detection and translation using LSTM, mediapipe and websockets.

People often find it difficult to learn or interpret sign language as it uses various hand gestures and movements and as the sign language is very vast it is difficult to recall and remember each sign. There are less video calling platforms or browser extensions present that may support sign language. Also currently a solution that people rely on for video conferencing is a human sign language interpreter but this interpreter might not always be available nor it may offer faster and seamless communication throughout the video call.

CoSign: Sign Language Translator Extension is an application developed to support American Sign Language detection and translation to make communication easy on virtual meeting platforms. It makes use of an LSTM trained model which uses mediapipe for holistic extraction of key points from hands and face for each word which is integrated with the browser extension where frames from the Google Meet are extracted and sent to the server where with the help of the trained model predictions of the actions are made and again sent back to the user displaying the detected sign language for the users enabling the extension.

3.2 Objective: -

- Easy communication between the hearing-speaking impaired and the unimpaired on video calling platforms.
- Accurate sign to text conversion.
- To bridge the communication gap between hearing/speaking impaired people and abled people.
- To develop an accurate machine learning model for predicting the sign gestures made throughout the video call in real time with reduced latency.
-

The application aids in providing a helping hand for hearing and speech impaired to communicate easily with the rest of the world on video calling platforms using sign language without having to rely on a human sign language translator as well as reducing the delay caused by different groups of people trying to communicate through two different types of language modes.

Signing gestures made by the signers must be converted accurately to text for non-signers to understand without interrupting the flow of the online meeting simultaneously understanding what the person signing is trying to communicate.

To bridge the communication gap between hearing/speaking impaired people and abled people who do not understand sign language or take time to comprehend the signs.

To develop an accurate machine learning model for predicting the sign gestures made throughout the video call in real time with reduced latency. Efficiently combining machine learning model with the created extension and ensuring it is collecting the frames from Google Meet of only the person who is signing and not others.

3.3 Scope: -

- Develop a convenient, compatible and user-friendly web extension that will work on top of video calling platforms to aid with sign language translations.
- A self-explanatory and easy to use User Interface.
- Sign to text translation which will detect signs via webcam and display corresponding text.

Chapter 4

Proposed System Architecture

4.1 Description about Proposed System:

The dataset was prepared for the model by using OpenCV which helped in accessing the camera of our device. Once the camera access was given, each word was signed 30 times and while doing so Mediapipe was constantly collecting key points for each frame and converting them into a numpy array. It created 30 such numpy arrays for each folder of each word. Later the dataset was used for training the model and saving it in an action.h5 file. By using OpenCV predictions were made to check the accuracy of the model.

After receiving a desired accuracy, the model was integrated with the extension. In the extension the non-impaired user will enable the extension and share his screen so that in the extension window he would be able to translate the sign gestures made by the impaired person. To make this possible the Screen Capture API was used to grab the frames of the ongoing Google Meet. Image Capture API was used to send the frames to the Websocket Server where Mediapipe was used again to plot the key points and then once the key points were plotted the LSTM model that was earlier integrated with the extension detects the action and displays it on the browser.

4.2 Diagrams:

4.2.1 Architecture Diagram:

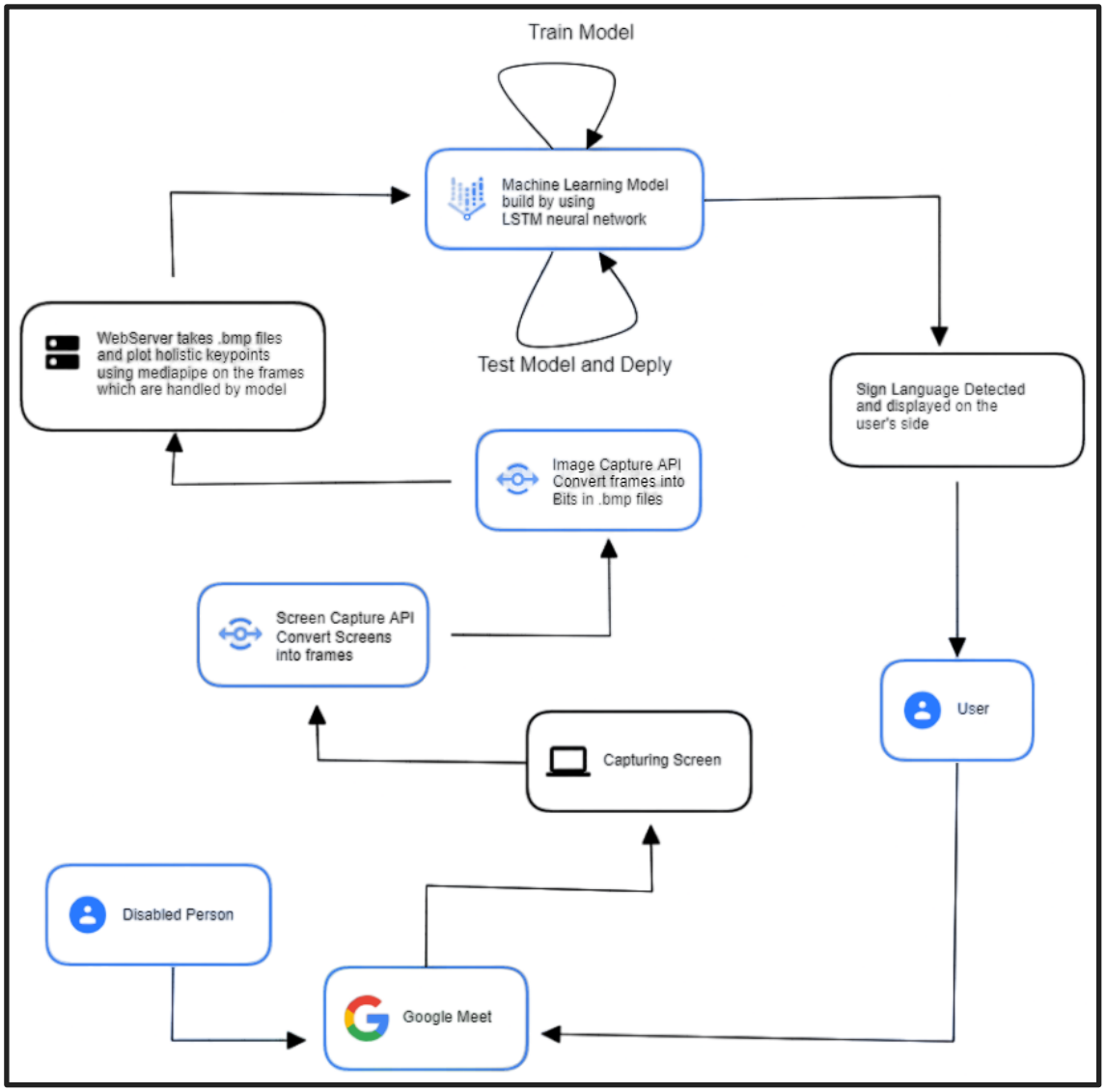
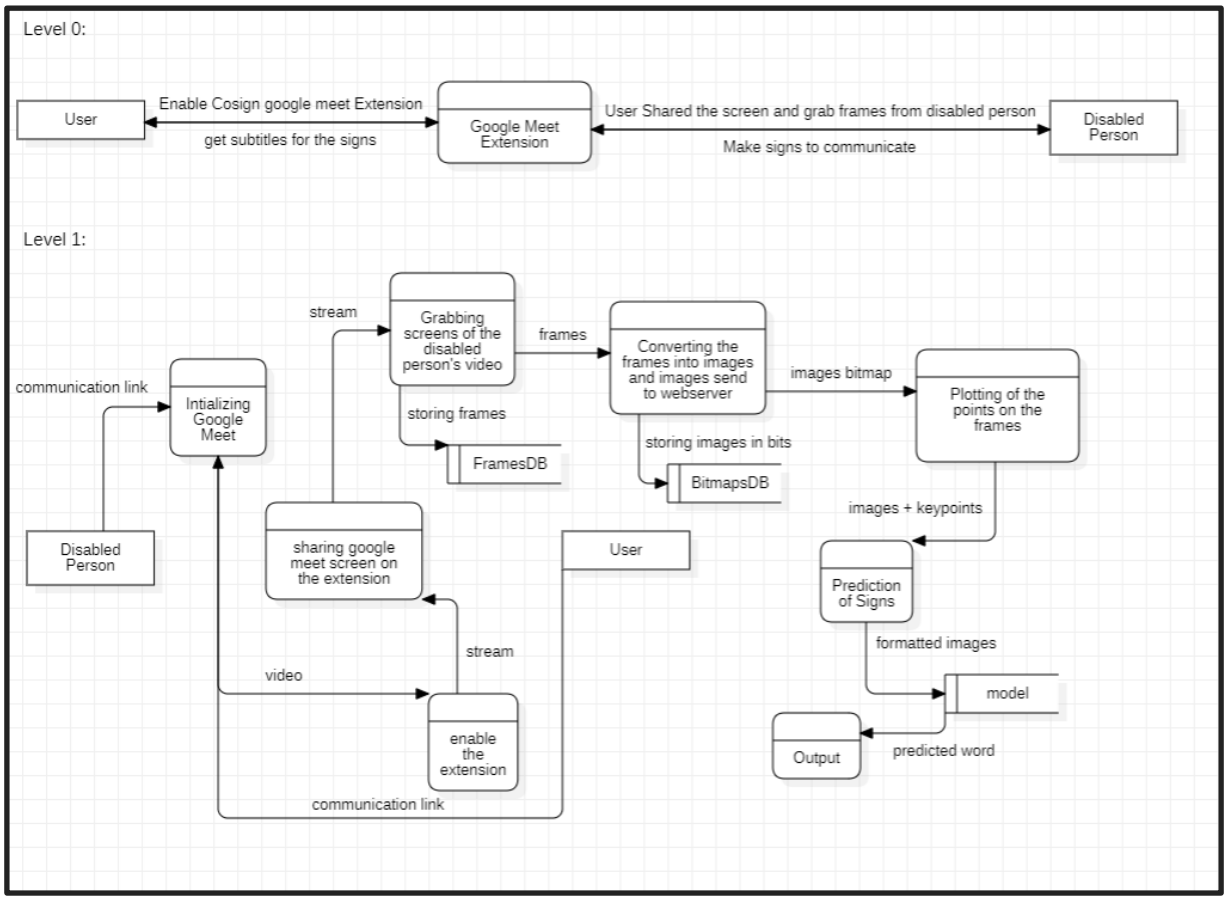


Figure 4.2.1: Architecture Diagram

An Architectural diagram is a visual representation that maps out the physical implementation of a software system. It shows the user's ability to use the entire application and the working of the application. The user and the disabled person both join the Google Meet, where the CoSign extension is enabled. The Screen Capture API then converts screens into frames which are then converted into Bits in .bmp files by Image Capture API. The WebServer then takes these files and plots holistic keypoints using mediapipes. The LSTM model then detects the Sign and displays it to the User.

4.2.2 Data Flow Diagram:



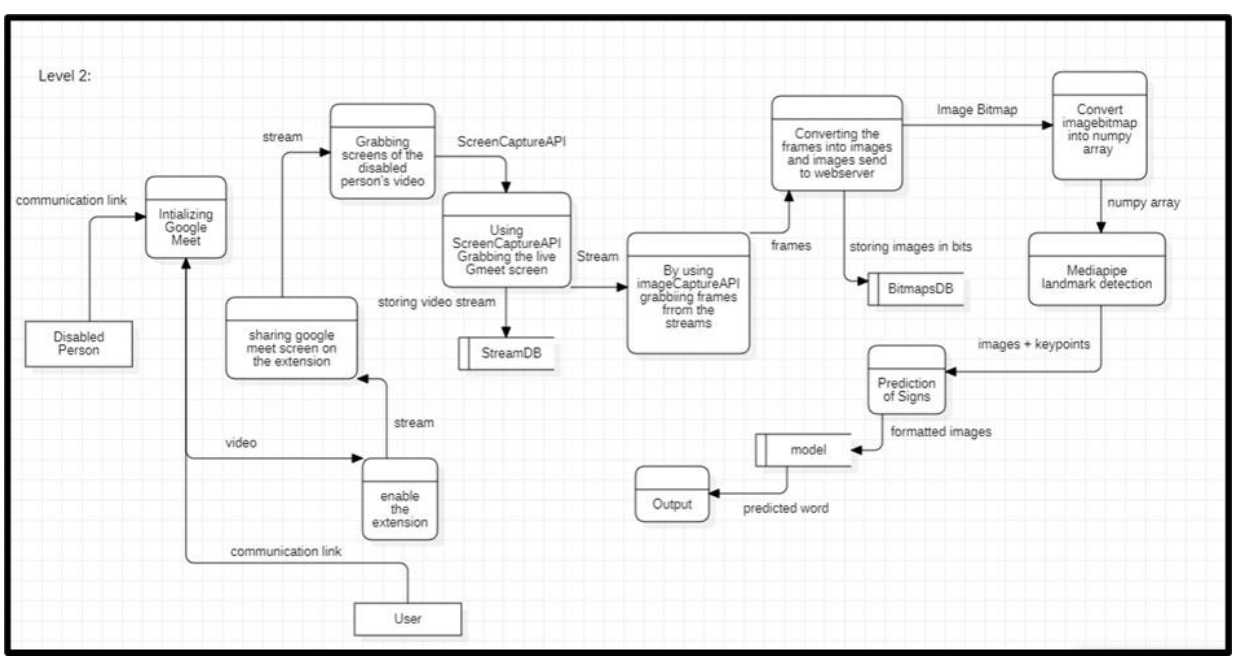


Figure 4.2.2: Data Flow Diagram

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles, and arrows, plus short text labels, to show data inputs, outputs, storage points, and the routes between each destination. Data flowcharts can range from simple, even hand-drawn process overviews, to in-depth, multi-level DFDs that dig progressively deeper into how the data is handled. They can be used to analyse an existing system or model a new one. The data flow diagram gives a clear picture of the project and the movement of the data from one object to another object as detailed information about the process and functionalities available in the project.

4.2.3 Flow Diagram of Model:

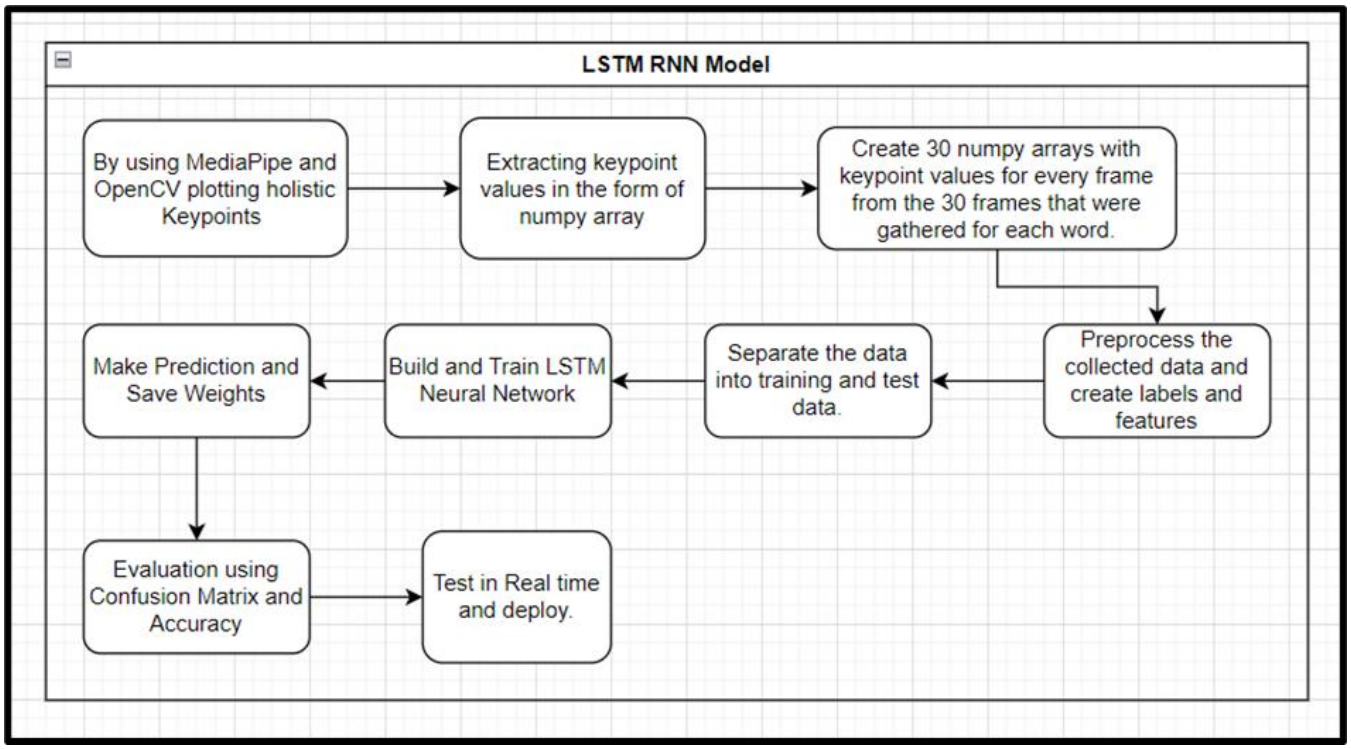


Figure 4.2.3: Flow Diagram of Model

Machine Learning architecture is the number of layers or the way that a machine learning algorithm solves a problem. CNN (Convolution Neural Networks) are used for Image Processing because they are designed for that and performs well, but the same architecture would not work for a problem where you need to keep track of previous events. For that kind of problem we use RNN (Recurrent Neural Networks). In this project of building browser extension for sign language translation we have used LSTM (Long Short Term Memory) Neural Network for building machine learning model. The architecture diagram above provides detailed information about the processes involved in creating a machine learning model. Building a model typically happens in his four phases: collecting datasets, separating data into training and testing data, applying LSTM neural networks to training data, and running tests on test data. The machine learning model was later saved in a .h5 file and used for further processing.

4.2.4 Use Case Diagram:

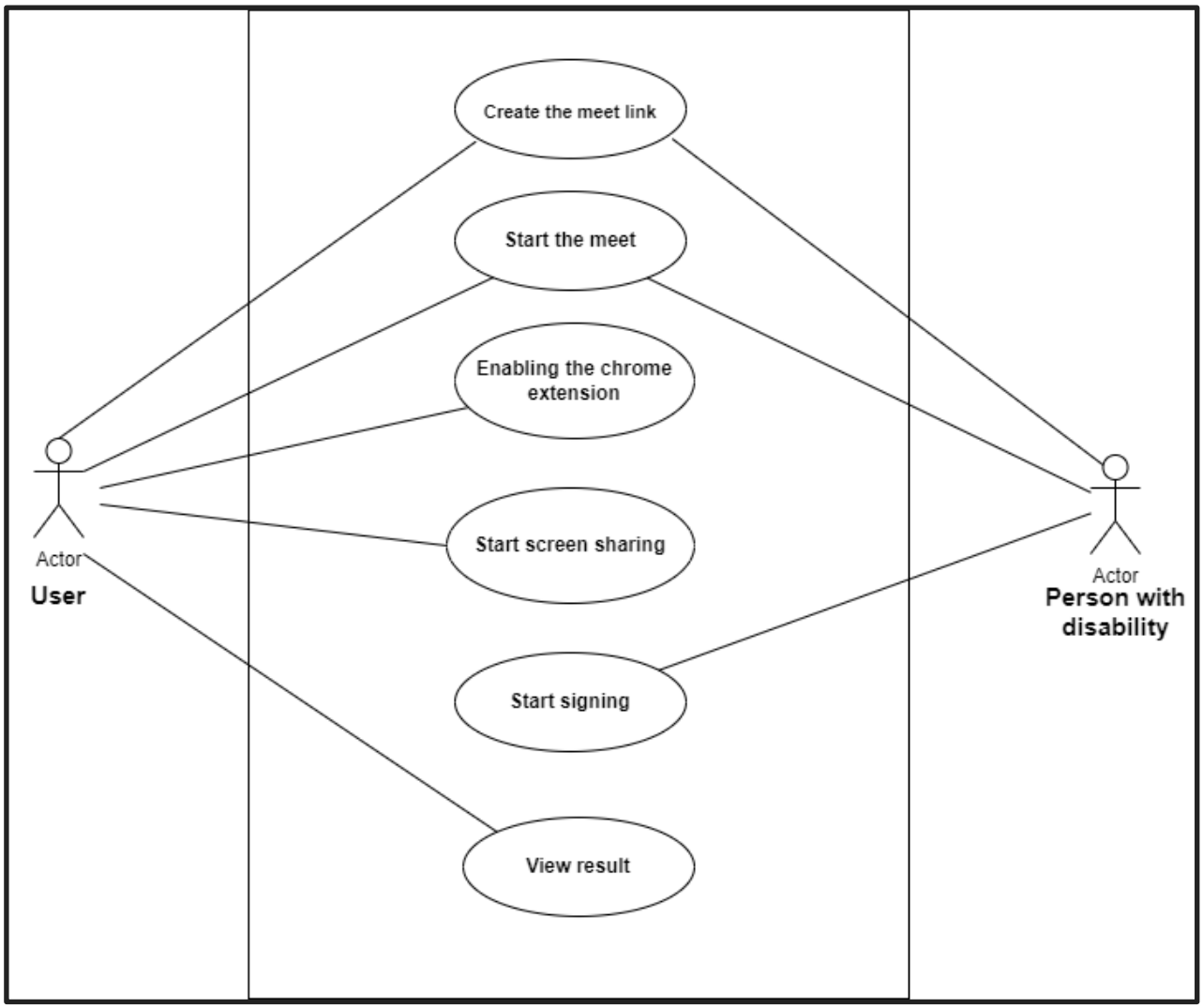


Figure 4.2.4: Use Case Diagram

A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses. Here the actors are the User and the Person with Disability and the overview of the process and functionalities are mentioned in the form of round-shaped diagrams. The use case diagram terminologies like include, exclude and generalization are properly used in the above use case diagram.

4.2.5 Sequence Diagram:

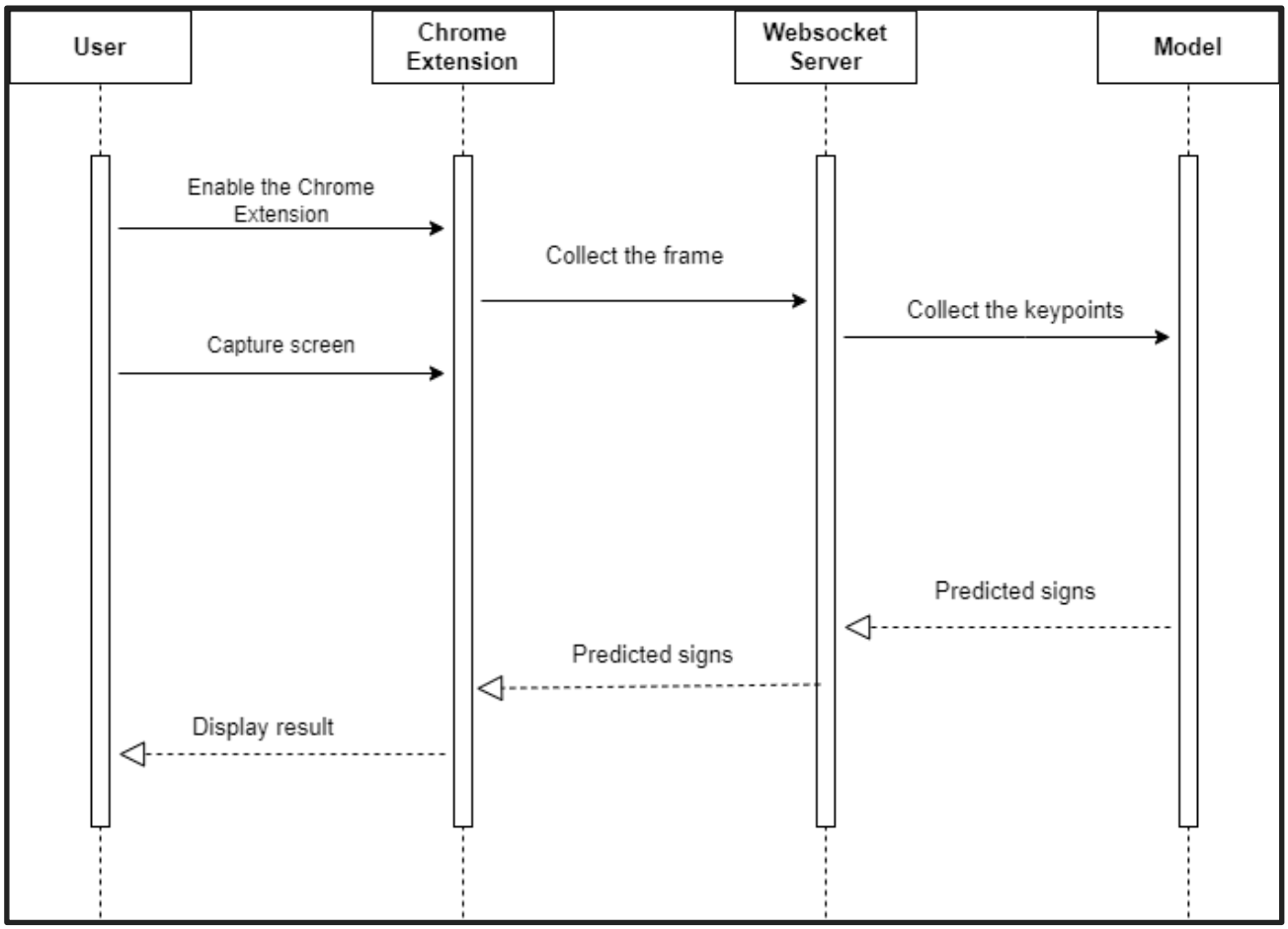


Figure 4.2.5: Sequence Flow Diagram

A sequence diagram simply depicts the interaction between objects in sequential order i.e., the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems. The above sequence diagram talks about the order in which happenings are taking place, and how the objects and their functions are dependent on each other. In the above diagram, objects are shown in the form of a rectangle shape whereas functionalities are represented in a flowing manner by using solid and dotted arrows.

4.2.6 Activity Diagram:

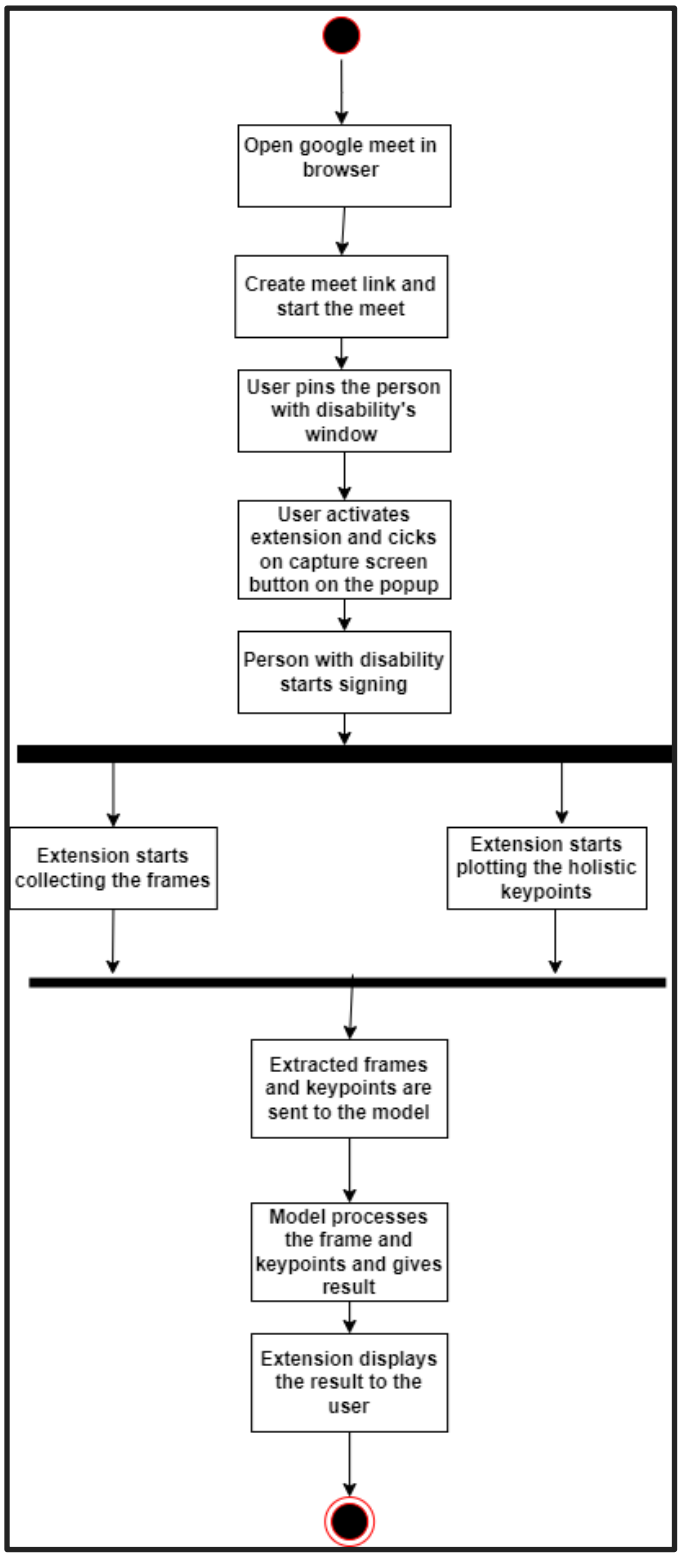
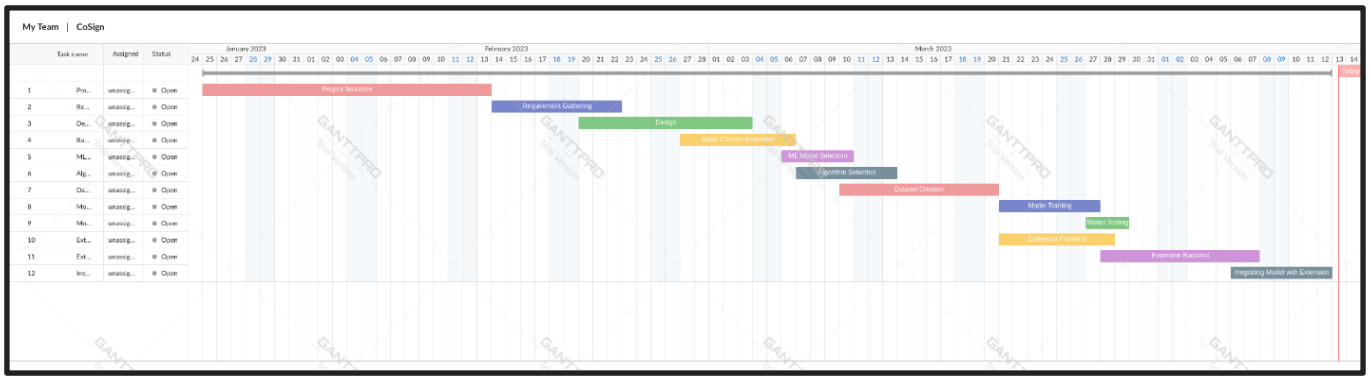


Figure 4.2.6: Activity Diagram

The activity diagram is another important diagram in UML to describe the dynamic aspects of the system. An activity diagram is a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all types of flow control by using different elements such as fork, join, etc. The activity diagram represents the entire work of the application in the flow schema and illustrates each component of the project simply so that the user or the student will be able to understand the whole work of the project.

Chapter 5

Project Planning



	Task name	Assigned	Status
1	Project Selection	unassigned	Done
2	Requirement G...	unassigned	Done
3	Design	unassigned	Done
4	Basic Chrome E...	unassigned	Done
5	ML Model Sele...	unassigned	Done
6	Algorithm Selec...	unassigned	Done
7	Dataset Creation	unassigned	Done
8	Model Training	unassigned	Done
9	Model Testing	unassigned	Done
10	Extension Front...	unassigned	Done
11	Extension Back...	unassigned	Done
12	Inegrating Mod...	unassigned	Done

Figure 5: Gantt Chart

Chapter 6

Experimental Setup

- **Software Requirements: -**

1. **HTML5**:

   HTML5 can be used in browser extensions to create user interfaces and interactive elements. Browser extensions allow developers to customize and enhance the functionality of web browsers, and HTML5 can be a useful tool in building these extensions.

   One way to use HTML5 in browser extensions is to create popup windows that display content and interact with the user. The popup window can be created using HTML, CSS, and JavaScript, and can be triggered by a button or other element on the extension's toolbar.

   In addition to popup windows, HTML5 can also be used to create options pages, settings menus, and other user interfaces for browser extensions. These elements can be designed and styled using HTML and CSS, and can be made interactive using JavaScript.

2. **CSS3**:

   CSS3 can be used in browser extensions to style and enhance the appearance of user interfaces and content. CSS3 offers a wide range of styling options, including layout, typography, color, animation, and more, which can be used to create visually appealing and engaging user interfaces.

   CSS3 can also be used to style other elements of the extension, such as settings menus, options pages, and toolbars. CSS3 features such as transitions and animations can also be used to create more dynamic and interactive user interfaces.

3. **JavaScript**:

JavaScript is an essential part of building browser extensions, as it enables developers to add dynamic functionality and interactivity to the extension's user interface and content. JavaScript can be used in many ways in browser extensions, such as:

Manipulating the DOM: JavaScript can be used to interact with the Document Object Model (DOM) of a web page or extension's content. For example, JavaScript can be used to dynamically modify the contents of a popup window, respond to user input, or manipulate HTML and CSS elements.

Interacting with APIs: Many browser APIs and web services offer JavaScript APIs that can be used in browser extensions. For example, the chrome.storage API can be used to store and retrieve data in the extension, while the chrome.tabs API can be used to interact with tabs and windows in the browser.

Event handling: JavaScript can be used to handle events, such as clicks, keyboard input, and page loads. For example, the chrome.browserAction.onClicked event can be used to handle clicks on the extension icon in the toolbar.

Content scripts: Content scripts are JavaScript files that run in the context of a web page, and can be used to modify the page's behavior or content. Content scripts can be used to interact with the DOM of the page, modify its styles or layout, or inject new elements into the page.

4. **Python-Tensorflow:**

The TensorFlow library is an open-source software library developed by Google Brain Team for numerical computation and machine learning. It is one of the most widely used deep learning libraries in the world, offering a wide range of tools, functions, and capabilities for building and training neural networks.

The TensorFlow library allows developers and researchers to create and train a wide variety of deep learning models, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and deep belief networks (DBNs). These models can

be used for a variety of applications, including image and speech recognition, natural language processing, and more.

Some of the key features of the TensorFlow library include:

1. High-level APIs that simplify the process of building and training neural networks.
2. Distributed computing capabilities that allow large-scale training of models across multiple devices and servers.
3. Support for a wide range of platforms, including desktops, mobile devices, and cloud servers.
4. Integration with other popular deep learning frameworks and tools, such as Keras, PyTorch, and TensorFlow.js.

To use the TensorFlow library, you will need to install it on your computer or server. This can be done using pip, a package installer for Python. Once installed, you can begin building and training your own deep learning models using the TensorFlow API. There are also many tutorials and resources available online to help you get started with TensorFlow.

5. Jupyter-Notebook:

Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. It supports multiple programming languages, including Python, R, and Julia, and can be used for a wide range of tasks, including data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and more.

The Jupyter Notebook interface consists of a web browser-based user interface, where you can write and execute code in cells, which are containers that can hold code, markdown text, or raw text. The output of the code is displayed below each cell, allowing you to see the results of your code in real-time.

Jupyter Notebook has become a popular tool for data science and scientific computing because it allows you to combine code and text in a single document, making it easier

to communicate your work and share it with others. You can also install and use various libraries and packages within Jupyter Notebook, such as NumPy, Pandas, and Matplotlib, which makes it a powerful tool for data analysis and visualization.

To get started with Jupyter Notebook, you need to install it on your computer or server, which can be done using pip, a package installer for Python. Once installed, you can launch Jupyter Notebook from the command line and start creating and executing code in cells. There are also many tutorials and resources available online to help you learn how to use Jupyter Notebook effectively.

6. **Mediapipe:**

MediaPipe is an open-source framework for building cross-platform multimodal machine learning applications. It was developed by Google and is used for a wide range of tasks, including computer vision, audio processing, and natural language processing.

MediaPipe provides a set of pre-built tools and models for developers and researchers to build complex pipelines that can take input from various sources, such as camera feeds, microphones, and sensors, and process them in real-time. The framework supports a wide range of platforms, including desktops, mobile devices, and edge devices, making it suitable for a variety of applications.

Some of the key features of MediaPipe include:

1. A flexible and modular architecture that allows developers to build custom pipelines by combining pre-built components.
2. A large set of pre-built models and tools for computer vision, audio processing, and natural language processing.
3. Support for a wide range of platforms and devices, including mobile phones, web browsers, and microcontrollers.
4. Real-time performance, enabling the development of applications that can process data in real-time.

MediaPipe is built using the C++ programming language, but also provides APIs for Python and JavaScript, making it accessible to a wide range of developers. To get started

with MediaPipe, you can download the framework from the official website and explore the pre-built models and tools. There are also many tutorials and resources available online to help you learn how to use MediaPipe effectively.

7. **OpenCV:**

OpenCV (Open Source Computer Vision) is an open-source library of programming functions mainly aimed at real-time computer vision. It was initially developed by Intel and is now maintained by the OpenCV Foundation. The library provides a comprehensive set of functions for image and video processing, including object detection and recognition, feature extraction, motion analysis, and more.

OpenCV is widely used in various applications, including robotics, surveillance systems, automotive driver assistance, medical image analysis, and augmented reality. It supports multiple programming languages, including C++, Python, and Java, and can be run on multiple platforms, including Windows, Linux, and macOS.

Some of the key features of OpenCV include:

1. Support for a wide range of image and video formats.
2. Built-in support for machine learning and deep learning algorithms.
3. A large set of pre-built functions for image and video processing, including filtering, segmentation, and feature detection.
4. Real-time performance, enabling the development of applications that can process data in real-time.
5. Integration with other popular libraries and frameworks, such as TensorFlow and PyTorch.

To use OpenCV, you need to install it on your computer or server, which can be done using pip, a package installer for Python. Once installed, you can use the OpenCV API to build and execute image and video processing algorithms. There are also many tutorials and resources available online to help you learn how to use OpenCV effectively.

- **Hardware Requirements: -**
    1. **CPU**:
        - For small to medium-sized datasets and less complex models, at least 4 cores and a clock speed of 2.5 GHz or higher may be sufficient.
        - For larger datasets and more complex models, a CPU with 8 or more cores and a clock speed of 3.0 GHz or higher
    2. **GPU**:

        For our machine learning projects, a GPU with at least 8 GB of memory is recommended. For if project gets more complex p, a GPU with 16 GB or more of memory may be required. The type of GPU also matters, and the most common choices for machine learning projects are NVIDIA GeForce, Titan, and Quadro series GPUs.

    3. **RAM**:

        For our machine learning projects, at least 8-16 GB of RAM is recommended, and the dataset should be stored on a fast solid-state drive (SSD) for optimal performance

    4. **STORAGE**:

        For small to medium-sized datasets, a storage capacity of 500 GB to 1 TB (SSD) may be sufficient. However, for larger datasets, such as those used in image or video processing, a storage capacity of 2 TB (SSD) or more may be required.

    5. **OS**:

        Windows, MacOS, Linux Distributions, etc.

Chapter 7

Implementation Details

7.1 Collecting Data:

Preparing the dataset by using OpenCV to capture frames from webcam. Using Mediapipe holistic to extract and plot key points. Through these once the webcam is accessed, sign 30 times for each word. Signing once for a particular word will generate 30 numpy arrays of the frame which consist of the key points for that sign. Hence, 30 signs for a single word * each sign generating 30 numpy arrays. Same procedure is repeated for the rest of the words and this completes our creation of data.

7.2 Preprocessing Data and Training:

Tensorflow keras is used to label the data. Sklearn is used to perform train-test-split where we are assigning 5% of data to testing. Tensorflow keras is used to build and train the LSTM Neural Network.

7.3 Chrome Extension Client:

Chrome Extension that works on the browser. User can enable the extension and click on the Capture Screen which will be displayed on the extension popup. Screen Capture API is used to fetch frames from the Google Meet screen. Image Capture API is used to send the images that are displayed on the extension popup to the server. The data is sent to the server via Websocket which will use the integrated trained model to preprocess the sign and sent to the browser where the detected sign will be displayed to the user.

7.4 Websocket Server:

Frames from the Google Meet screen are received via websocket. Mediapipe plots the key points on the frame and the integrated model is used to detect and recognize the sign. The detected sign is sent to the browser via websocket where the user is able to see the predicted text for the respective sign.

7.5 Predict and Display:

Frames and key points are passed through the model and the resultant output is displayed to the client.

Chapter 8

Result

M1: Collecting Data

```
cap = cv2.VideoCapture(0)
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
    for action in actions:
        for sequence in range(no_sequences):
            for frame_num in range(sequence_length):
                ret, frame = cap.read()
                image, results = mediapipe_detection(frame, holistic)
                draw_styled_landmarks(image, results)
                if frame_num == 0:
                    cv2.putText(image, 'STARTING COLLECTION', (120,200),
                                cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255, 0), 4, cv2.LINE_AA)
                    cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence), (15,12),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                    cv2.imshow('OpenCV Feed', image)
                    cv2.waitKey(2000)
                else:
                    cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence), (15,12),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                    cv2.imshow('OpenCV Feed', image)
                keypoints = extract_keypoints(results)
                npy_path = os.path.join(DATA_PATH, action, str(sequence), str(frame_num))
                np.save(npy_path, keypoints)
                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break
    cap.release()
    cv2.destroyAllWindows()
```

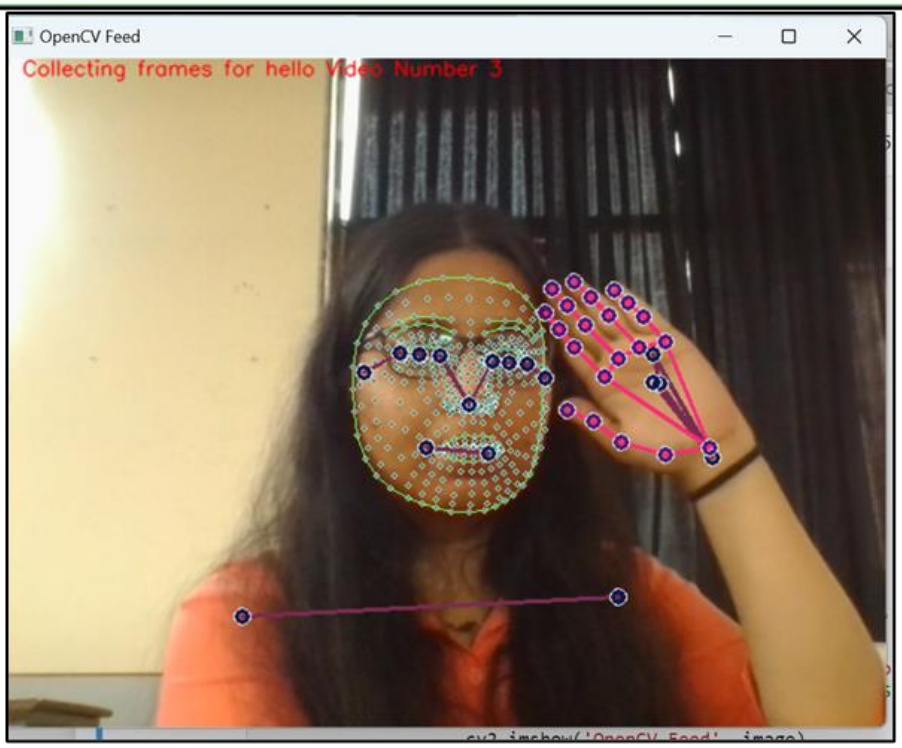


Figure 8.1: Collecting Data

Collecting data in the form of numpy arrays by storing
holistic keypoint values generated with the help of Mediapipe and OpenCv

M2: Preprocessing Data and Training of Model:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import TensorBoard
```

```
log_dir = os.path.join('Logs')
tb_callback = TensorBoard(log_dir=log_dir)
```

```
model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(30,1662)))
model.add(LSTM(128, return_sequences=True, activation='relu'))
model.add(LSTM(64, return_sequences=False, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(actions.shape[0], activation='softmax'))
```

```
res = [.7, 0.2, 0.1]
```

```
actions[np.argmax(res)]
```

```
model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])
```

```
model.fit(X_train, y_train, epochs=2000, callbacks=[tb_callback])
```

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_6 (LSTM)                (None, 30, 64)            442112
_________________________________________________________________
lstm_7 (LSTM)                (None, 30, 128)           98816
_________________________________________________________________
lstm_8 (LSTM)                (None, 64)                49408
_________________________________________________________________
dense_6 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_7 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_8 (Dense)              (None, 3)                 99
=================================================================
Total params: 596,675
Trainable params: 596,675
Non-trainable params: 0
_________________________________________________________________
```

```
res = model.predict(X_test)
```

```
actions[np.argmax(res[4])]
```

```
'hello'
```

```
actions[np.argmax(y_test[4])]
```

```
'hello'
```

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

Figure 8.2: Preprocessing Data and Training

We processed and cleaned the data collected during the data collection phase. Using the LSTM Neural Network, we have trained the model required for sign language recognition.

M3: Chrome Extension Client

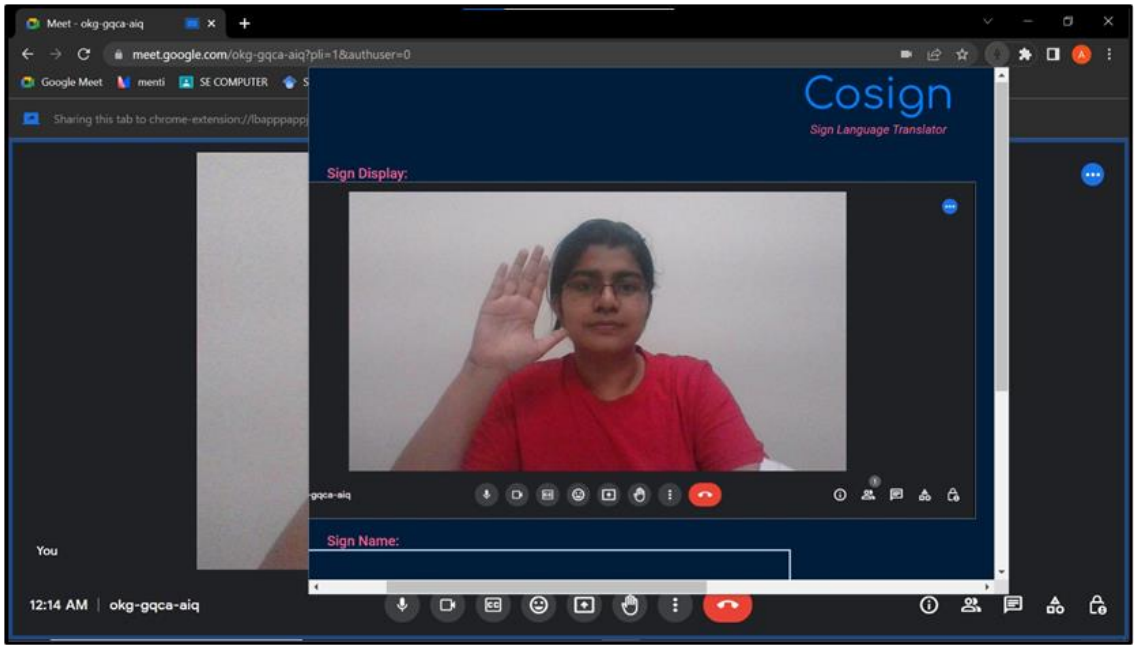


Figure 8.3: Chrome Extension Client

It represents the frontend part where the user is able to make use of Cosign extension where he can see signs made by Signer on the "Sign Display" section by sharing his gmeet screen.

M4: Websocket Server

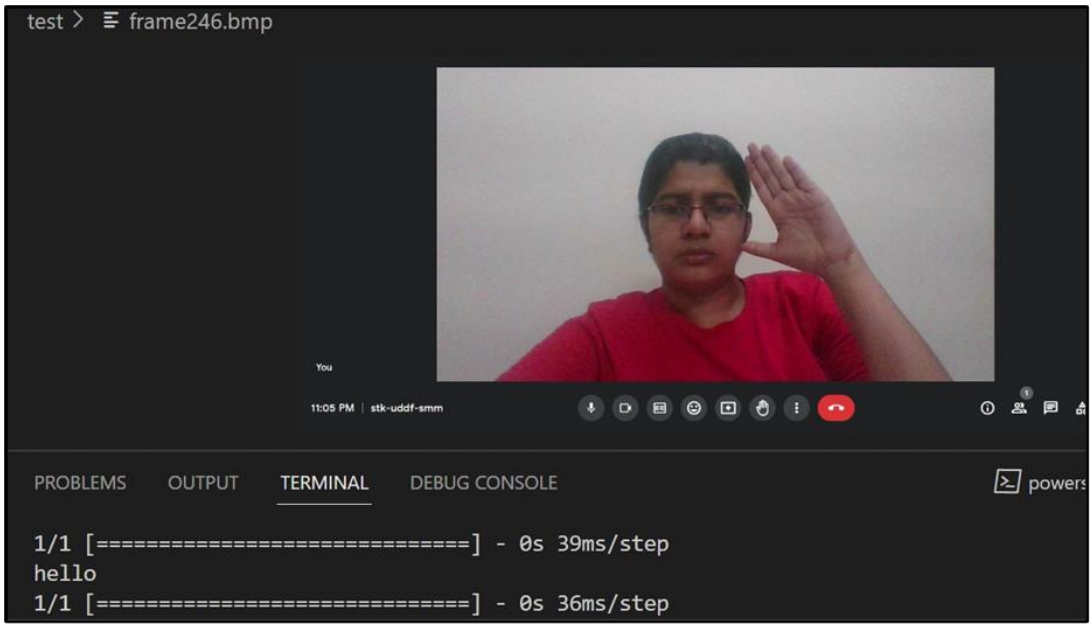


Figure 8.4: Websocket Server

Websocket server takes the images from frames which are converted from streams of videos with the help of imageCapture API and Processed them for the model.

M5: Predict and Display:

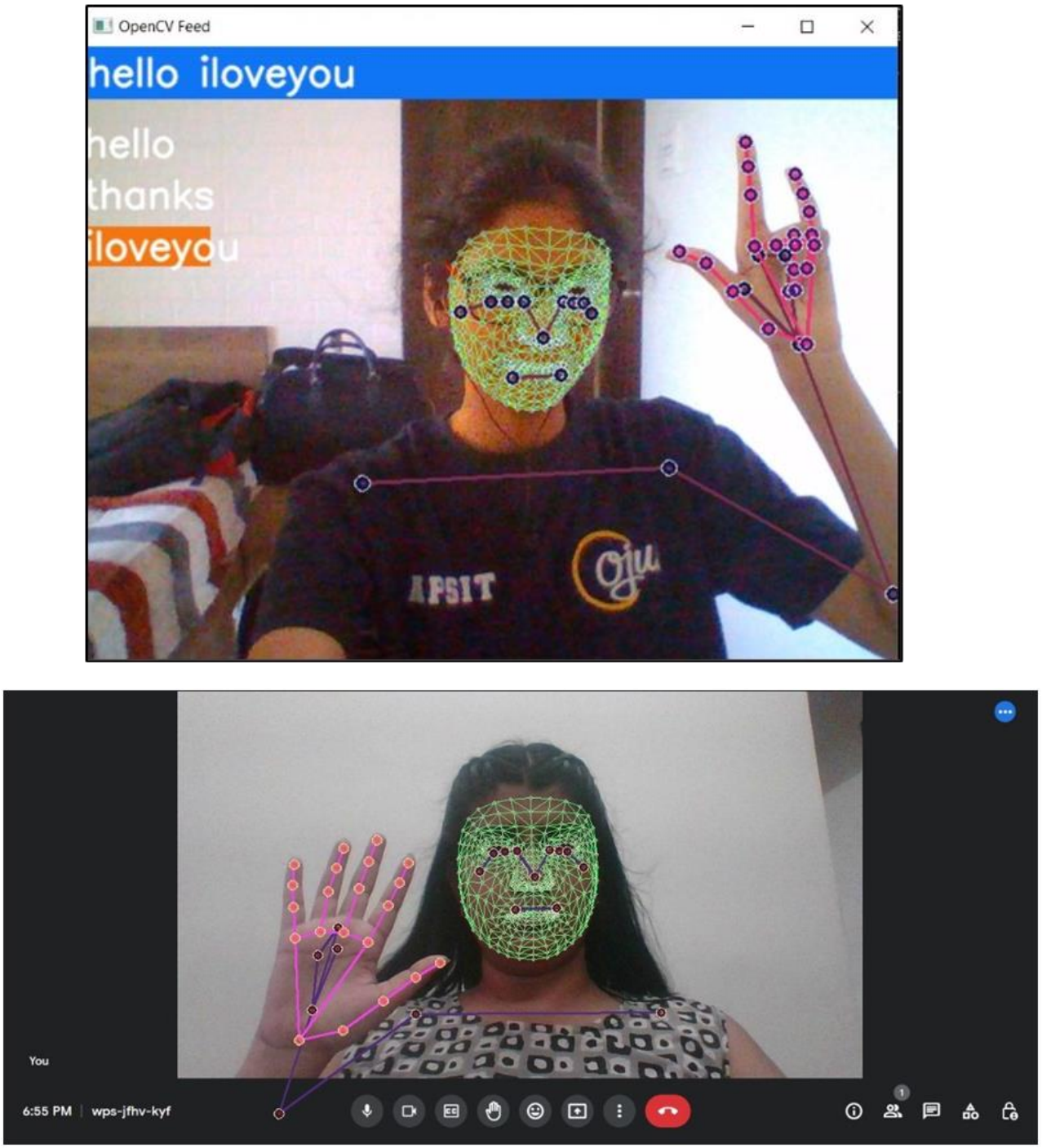


Figure 8.5: Predict and Display

ImageBitmap converted into the numpy arrays and mediapipe landmark detection is performed using this numpy arrays and sends to the model for prediction. After prediction required output for a particular sign is generated at the “Sign Text” section of the Cosign browser extension.

Chapter 9

Conclusion

The issues that arise from communication between the hearing, speaking majority and non-hearing, non-speaking majority especially during online virtual meetings can be easily resolved by enabling the CoSign: Sign Language Translator Extension without the need for any other complex solutions for example requirement of a human interpreter or the non-signers learning and remembering all the signs in American Sign Language.

By detecting and translating sign language in real-time, the extension can bridge communication barriers and provide a more inclusive and engaging meeting experience. With the help of advanced computer vision and machine learning technologies, the extension can accurately recognize and interpret different sign language gestures and expressions. Additionally, the use of WebSockets allows for seamless communication and integration with the Google Meet platform.

The hard-of-hearing and mute individual needs to keep his Google Meet video on throughout the entire video conference and the user will enable the extension and share the screen to fetch the Google Meet frames and these frames will be sent to the server where predictions will be made and sent as a response to the browser which will show the predicted word to the user.

Overall, this extension has the potential to make online meetings more accessible and inclusive for people with hearing and speech disabilities and help them communicate with non-disabled individuals without worrying about not being unde

Chapter 10

References

[1] Bantupalli, K. and Xie, Y., 2018, December. American sign language recognition using deep learning and computer vision. In *2018 IEEE International Conference on Big Data (Big Data)* (pp. 4896-4899). IEEE.

[2] Yuan, T., Sah, S., Ananthanarayana, T., Zhang, C., Bhat, A., Gandhi, S. and Ptucha, R., 2019, May. Large scale sign language interpretation. In *2019 14th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019)* (pp. 1-5). IEEE.

[3] Zheng, L., Liang, B. and Jiang, A., 2017, November. Recent advances of deep learning for sign language recognition. In *2017 International Conference on Digital Image Computing: Techniques and Applications (DICTA)* (pp. 1-7). IEEE.

[4] Rastgoo, R., Kiani, K. and Escalera, S., 2021. Sign language recognition: A deep survey. *Expert Systems with Applications*, *164*, p.113794.

[5] Kushalnagar, R.S. and Vogler, C., 2020, October. Teleconference accessibility and guidelines for deaf and hard of hearing users. In *Proceedings of the 22nd International ACM SIGACCESS Conference on Computers and Accessibility* (pp. 1-6).