Major Project Report
CSN 451

(Semester 22232)

Medical Dataset and Translation for Indian Sign Language (ISL)

Submitted by –

Akshit Jain	**19103059**
Sarthak Gaba	**19103045**
Dhruv Purwar	**19103064**
Kavya Malhotra	**19103086**

Under the Guidance of:

Dr. Shailendra Singh
Professor
CSE Department
Punjab Engineering College
(Deemed to be University)
Project Mentor

Prof. Sahil
Faculty
Punjab Engineering College
(Deemed to be University)
Project Mentor

Department of Computer Science and Engineering (CSE)
Punjab Engineering College, Chandigarh (Deemed to be University)
August 2022 to May 2023

DECLARATION

We hereby declare that the project carried out by our team from **August 2022 to May 2023** for the course Major Project under the Computer Science and Engineering Department, Punjab Engineering College as requirements of the **final year project** for the award of degree of B.Tech. Computer Science and Engineering, Punjab Engineering College, Chandigarh, under the guidance of **Dr. Shailendra Singh and Dr. Arun Singh Pundir, Computer Science and Engineering Department, Punjab Engineering College (Deemed to be University), Chandigarh** is an authentic record of our own work.

Akshit Jain
19103059

Sarthak Gaba
19103045

Dhruv Purwar
19103064

Kavya Malhotra
19103086

CERTIFICATE

This is to certify that this project report entitled **"Medical Dataset and Recognition for Indian Sign Language"** submitted to **Punjab Engineering College (Deemed to be University), Chandigarh,** is a trusted record of work done by **"Akshit Jain(19103059), Sarthak Gaba(19103045), Dhruv Purwar(19103064), Kavya Malhotra(19103086)"** under the supervision of **Dr. Shailendra Singh and Dr. Arun Singh Pundir, Faculty, Computer Science and Engineering Department, Punjab Engineering College (Deemed to be University), Chandigarh** in fulfilment of the requirements as a part of Major Project for the award of 06 credits in Semester 8 of the degree of Bachelor of Technology in Computer Science and Engineering.

Certified that the above statement made by the students is correct to the best of my knowledge and belief.

Prof. Sahil
Assistant Professor

Dr. Shailendra Singh
Professor, CSE

Date: 24/05/2023

ACKNOWLEDGEMENT

This is to acknowledge all those without whom this project would not have been reality. Firstly, we would like to thank our Computer Science and Engineering department, Punjab Engineering College for providing us with this great learning opportunity.

A project is a bridge between theoretical and practical learning and with this opportunity we were able to further strengthen our understanding of different aspects of Computer Science.

We take this opportunity to express our profound gratitude and deep regards to my teachers Dr. Shailendra Singh, and Prof. Sahil, Faculty, Computer Science and Engineering Department, Punjab Engineering College (Deemed to be University), Chandigarh for their exemplary guidance, monitoring and constant encouragement throughout the course of this project.

We would also like to thank the panel members of our major project evaluation for their continuous guidance and showing us the areas to improve.

- Akshit Jain, Sarthak Gaba, Dhruv Purwar, Kavya Malhotra

ABSTRACT

Sign Language serves as a non-verbal communication system utilized by individuals to visually convey meaning through sign patterns. In India, a substantial portion of the hearing and speech impaired community relies on Indian Sign Language (ISL) as their primary mode of communication.

ISL is a comprehensive language with its own unique grammar, syntax, vocabulary, and linguistic attributes. With over 5 million deaf individuals in India using ISL, there is currently a lack of publicly available datasets for evaluating Sign Language Recognition (SLR) approaches. To address this gap, we present the Indian Lexicon Sign Language Medical Dataset (ISL Medical), specifically focused on medical terminology. ISL Medical comprises numerous videos featuring medical words, recorded by experienced signers to closely resemble natural conditions. Additionally, our project aims to provide a text/speech-to-sign language conversion feature and vice versa.

The recognition of sign language holds significance not only from a technical perspective but also in terms of its societal impact. By facilitating effective communication between healthcare providers and individuals who rely on ISL, this project has the potential to greatly enhance accessibility and inclusivity in healthcare settings. The development of ISL recognition systems and the availability of the ISL Medical dataset contribute to the advancement of assistive technologies and pave the way for improved healthcare services for the Indian deaf community.

Keywords: Sign Language, Indian Sign Language (ISL), Sign Language Recognition (SLR), ISL Medical dataset, deep learning, accessibility, inclusivity, assistive technologies, healthcare services.

TABLE OF CONTENTS

LIST OF FIGURES

List of Tables

CHAPTER 1

INTRODUCTION

Introduction

Communication is a fundamental aspect of human interaction, and language serves as the primary medium for expressing thoughts, ideas, and emotions. However, for individuals with hearing and speech impairments, traditional spoken languages may not be accessible. In such cases, sign languages emerge as powerful tools for visual communication. Sign languages are complete linguistic systems that utilize gestures, hand movements, facial expressions, and body postures to convey meaning. Among the diverse sign languages used worldwide, Indian Sign Language (ISL) is a prominent one, with its own unique grammar, syntax, vocabulary, and linguistic attributes. Recognizing the importance of effective communication for the deaf community, this project focuses on two crucial aspects: dataset creation and model development for ISL recognition.

1.1 Introduction

The Indian Sign Language Recognition and Medical Dataset is a significant resource that addresses the scarcity of labeled datasets for Indian Sign Language (ISL) recognition. It serves as a comprehensive collection of diverse ISL gestures and expressions, providing a strong foundation for accurate interpretation and recognition of ISL.

Recognizing and interpreting ISL poses unique challenges due to the complexity and variability of hand movements, facial expressions, and body postures involved in sign language communication. The dataset encompasses a wide range of these gestures, capturing the rich linguistic and cultural aspects of ISL. It enables researchers and developers to train and evaluate ISL recognition models, driving advancements in accuracy and effectiveness.

The dataset also includes a specialized component focusing on medical sign language. Medical sign language plays a vital role in healthcare settings, allowing deaf or hard-of-hearing individuals to effectively communicate with medical professionals. By incorporating medical sign language gestures and expressions, the dataset caters to the specific needs of healthcare contexts, ensuring accurate interpretation of medical conversations and instructions.

By leveraging the Indian Sign Language Recognition and Medical Dataset, researchers and developers can explore various recognition approaches, including deep learning models such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs). These models can be trained on the dataset to learn the intricate patterns and variations of ISL, enabling them to accurately recognize and interpret sign language gestures.

Accurate recognition of ISL is essential for facilitating effective communication and inclusion for individuals with hearing impairments. The Indian Sign Language Recognition and Medical Dataset plays a pivotal role in advancing ISL recognition research and development. It empowers researchers, data scientists, and developers to enhance the accuracy, efficiency, and accessibility of ISL recognition systems, ultimately fostering inclusivity and enabling deaf and hard-of-hearing individuals to communicate effectively in various domains, including healthcare.

1.1.1 Dataset Creation

To facilitate accurate and robust ISL recognition, a comprehensive dataset on medical data has been meticulously curated. We collaborated with Lions School for the Deaf, Sector 18, Chandigarh, to create a diverse collection of videos capturing ISL gestures related to medical terminology and concepts has been recorded. The dataset ensures authenticity and natural signing conditions by involving experienced signers during the recording process. It encompasses a wide range of medical words and phrases, empowering healthcare providers to communicate effectively with individuals who rely on ISL. The creation of this dataset aims to address the lack of publicly available resources specific to ISL and facilitate advancements in sign language recognition technology.

1.1.2 Model Creation

Deep learning techniques have revolutionized various fields, including computer vision and natural language processing. Leveraging these advancements, this project focuses on developing an accurate and robust system for ISL recognition. The model will be trained using the carefully curated ISL medical dataset, enabling it to learn the intricate gestures, movements, and expressions that constitute ISL.

By utilizing convolutional neural networks (CNNs) and recurrent neural networks (RNNs), the model will analyze and interpret the visual features of ISL gestures. The goal is to create a system that can accurately recognize and translate ISL gestures into corresponding text or speech, bridging the communication gap between the deaf community and the wider society.

Through the combined efforts of dataset creation and model development, this project aims to enhance accessibility and inclusivity for individuals who rely on ISL in medical contexts. By providing healthcare professionals with a reliable tool for ISL recognition, the project aspires to improve communication, healthcare delivery, and overall quality of life for the deaf community.

CHAPTER 2

BACKGROUND

2.1 PROBLEM DESCRIPTION

The problem at hand is the recognition of Indian Sign Language (ISL) gestures for effective communication with individuals who are deaf or have hearing impairments. Sign language serves as a vital mode of communication for the deaf community, enabling them to express their thoughts, needs, and emotions visually. However, existing solutions for ISL recognition are limited, and there is a need to develop more accurate and robust models to bridge the communication gap between the deaf and the hearing communities.

Currently, there is a scarcity of publicly available datasets specific to ISL, making it challenging to develop and evaluate ISL recognition approaches. Most existing solutions in sign language recognition primarily focus on widely used sign languages, such as American Sign Language (ASL) or British Sign Language (BSL). These solutions employ various techniques, including computer vision and machine learning algorithms, to interpret hand gestures, movements, and facial expressions.

However, these solutions do not cater specifically to the unique grammar, syntax, and vocabulary of ISL. As a result, their performance in accurately recognizing ISL gestures may be suboptimal. Moreover, the lack of comprehensive datasets for ISL further limits the development and evaluation of ISL recognition models.

2.2 OBJECTIVE

To address these challenges, our project aims to develop a robust system for Indian Sign Language (ISL) recognition in the context of medical communication. We have undertaken the task of creating a comprehensive ISL medical dataset in collaboration with a local school for the deaf. This dataset consists of videos capturing a wide range of ISL gestures related to medical terminology and concepts, recorded under natural signing conditions with experienced signers.

With the help of this extensive dataset, we will employ deep learning techniques, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), to train a model specifically tailored for ISL recognition. The model will learn to analyze the visual features of ISL gestures, including hand movements, facial expressions, and body postures, to accurately recognize and interpret the intended meaning. Our solution aims to bridge the communication gap in medical settings, enabling healthcare providers to effectively communicate with individuals who rely on ISL.

By developing an accurate and robust ISL recognition model, we seek to improve accessibility, inclusivity, and the overall quality of healthcare services for the deaf community. Our solution endeavors to empower healthcare professionals to communicate seamlessly with individuals using ISL, enhancing patient care and promoting equal access to healthcare information and services.

2.3 TECHNOLOGY USED

2.3.1 Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all majorplatforms, and can be freely distributed. Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debuga program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

2.3.2 MediaPipe

MediaPipe is an open-source framework developed by Google that offers a comprehensive set of cross-platform tools and libraries for building applications with perceptual capabilities, such as keypoints detection. Keypoints detection involves identifying specific points of interest on various body parts, enabling applications to understand and analyze human movements, facial expressions, and hand gestures.

In MediaPipe, keypoints detection is achieved using pre-trained machine learning models that are capable of accurately localizing these keypoints. The framework provides ready-to-use models and pipelines for face, pose, and hand keypoints detection, making it convenient for developers to integrate these capabilities into their applications.

- Face Keypoints Detection: MediaPipe offers a face detection and face landmarks model that can accurately detect and locate facial features, including key points such as the eyes, nose, mouth, and facial contours. By analyzing the position and movement of these keypoints, applications can perform tasks like facial expression analysis, emotion recognition, and augmented reality effects.

- Pose Keypoints Detection: MediaPipe's pose detection model allows the estimation of key points on a person's body, enabling the tracking of human pose and movements. The keypoints include positions on the head, neck, shoulders, elbows, wrists, hips, knees, and ankles. By leveraging this information, applications can enable fitness tracking, gesture recognition, virtual try-on experiences, and more.

- Hand Keypoints Detection: MediaPipe provides hand tracking and hand landmarks models for robust detection and tracking of hand keypoints. These keypoints represent the fingers, palm, and other landmarks on the hand. Applications can utilize this information for gesture-based control, sign language recognition, virtual object manipulation, and interactive experiences.

It's important to note that MediaPipe supports both 2D and 3D keypoints estimation. While 2D keypoints provide the (x, y) coordinates of the keypoints in the image plane, 3D keypoints additionally estimate the depth or distance information, allowing for more accurate spatial understanding.

2.3.3 RNN

A recurrent neural network (RNN) model, also known as a Sequential Model is a type of machine learning model designed to handle sequential data. It is particularly useful for tasks that involve processing and making predictions on sequences or time-series data.

Sequential models are called “sequential” because they are capable of capturing the sequential nature of the data they are trained on. They can process data points in a sequence, one at a time, while maintaining an internal state that retains information from previous data points. This internal state allows the model to learn and remember patterns and dependencies in the sequence.

Architecture:
The basic building block of a sequential model is the recurrent neural network (RNN). An RNN processes input data sequentially and maintains a hidden state that is updated at each time step. The hidden state serves as a memory or representation of the previous information seen by the model. The RNN architecture allows information to be propagated through time and capture dependencies between different elements in the sequence.

One popular type of RNN is the Long Short-Term Memory (LSTM) network. LSTMs address the vanishing gradient problem, which can occur when training traditional RNNs. The vanishing gradient problem refers to the issue of the gradients becoming very small during backpropagation, making it difficult for the model to learn long-term dependencies. LSTMs use a system of gating mechanisms (input gate, forget gate, and output gate) to selectively remember and forget information over time, allowing them to capture long-term dependencies effectively.

Another notable architecture in sequential models is the Transformer. Transformers were originally introduced for natural language processing (NLP) tasks but have been extended to handle other sequential data as well. Transformers rely on self-attention mechanisms to capture dependencies between different elements in a sequence. Instead of processing sequences sequentially, transformers can process all elements in parallel, making them computationally efficient. They have gained popularity due to their effectiveness in capturing long-range dependencies.

Training and Applications:

Sequential models are trained using a variant of backpropagation called backpropagation through time (BPTT). BPTT unfolds the recurrent connections of the model over time, creating a computational graph that can be used to compute gradients and update the model's parameters.

Sequential models have been successfully applied to a wide range of tasks, including:

1. Natural Language Processing (NLP): Tasks such as language 19oolean1919, machine translation, sentiment analysis, text generation, and named entity recognition.
2. Speech and Audio Processing: Tasks such as speech recognition, speech synthesis, speaker identification, and audio classification.
3. Time Series Analysis: Tasks such as time series forecasting, anomaly detection, and signal processing.
4. Gesture and Action Recognition: Tasks that involve recognizing and understanding human gestures and actions in videos.
5. Bioinformatics: Tasks such as protein structure prediction, DNA sequence analysis, and genomics.
6. Robotics and Autonomous Systems: Tasks such as robot control, motion planning, and sensor fusion.

Advancements and Future Directions:

Sequential models have seen significant advancements in recent years. Researchers have proposed various improvements and extensions to the basic RNN and LSTM architectures. Some notable developments include:

a) Gated Recurrent Units (GRUs): GRUs are an alternative to LSTMs that also address the vanishing gradient problem. They have a simpler architecture compared to LSTMs but have been shown to perform similarly in many tasks.

b) Variants of Transformers: Researchers have introduced variations of the Transformer architecture, such as the Transformer-XL, which addresses the limitation of the original Transformer in handling long-range dependencies.

c) Hybrid Architectures: Sequential models have been combined with other types of neural networks, such as convolutional neural networks (CNNs), to leverage their respective strengths in processing different types of data.

d) Reinforcement Learning: Sequential models have been combined with reinforcement learning algorithms to learn sequential decision-making policies.

In summary, sequential ML models, such as RNNs and LSTMs, are powerful tools for analyzing and predicting sequential data. They excel in tasks where the order and temporal dependencies of the data are crucial for accurate predictions. With ongoing research and advancements, sequential models continue to evolve and find applications in various domains.

2.3.4 LSTM

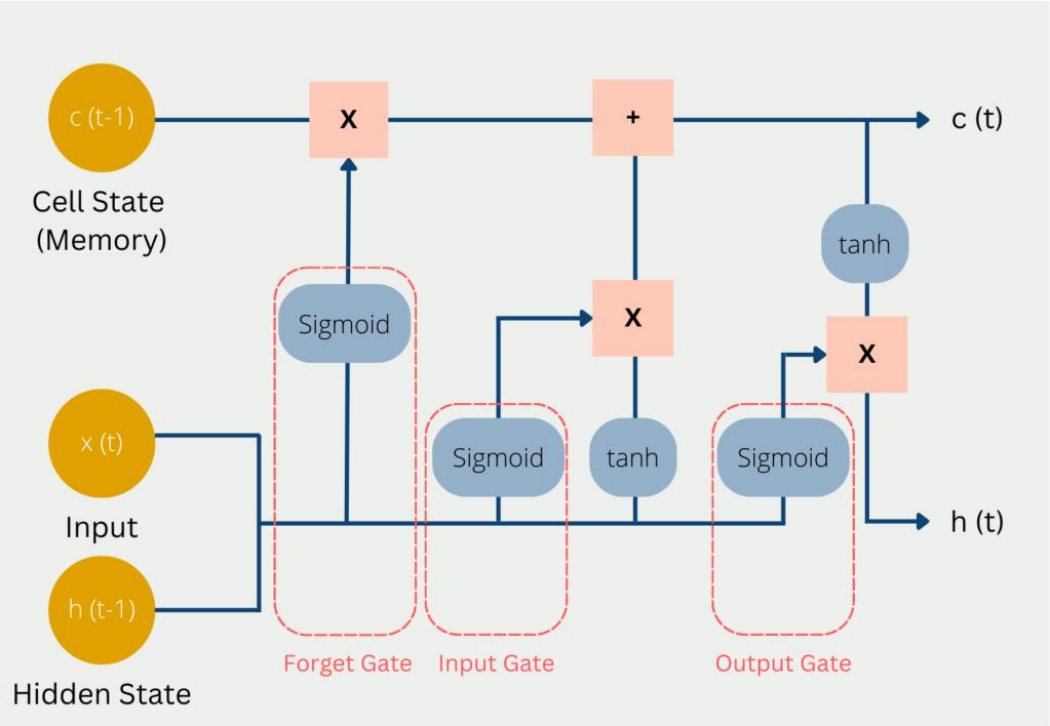

Figure 2: LSTM Architecture
(Source: https://databasecamp.de/en/ml/lstms)

LSTM is a type of recurrent neural network (RNN) architecture that is specifically designed to address the vanishing gradient problem, which can occur when training traditional RNNs. The vanishing gradient problem refers to the issue of the gradients becoming very small during backpropagation, making it difficult for the model to learn long-term dependencies in sequential data.

The key innovation of LSTM is its ability to selectively remember and forget information over time using a system of gating mechanisms. These mechanisms are built into the LSTM cells, which are the basic building blocks of the LSTM model.

Let's break down the components of an LSTM cell:

Fig 2: LSTM

- Cell State (Ct): The cell state represents the "memory" of the LSTM. It runs along the entire sequence, and information can flow through the cell state without much modification. It serves as a long-term memory and allows the LSTM to capture dependencies over longer time scales.

- Input Gate (i): The input gate determines how much of the new input should be stored in the cell state. It takes the input at the current time step and the previous hidden state as inputs and applies

a sigmoid activation function to produce values between 0 and 1. A value close to 0 means that the input should be ignored, while a value close to 1 means that the input should be stored.

- Forget Gate (f): The forget gate decides what information should be discarded from the cell state. It takes the input at the current time step and the previous hidden state as inputs and applies a sigmoid activation function. The output of the forget gate is multiplied element-wise with the previous cell state, allowing the LSTM to forget irrelevant information.

- Output Gate (o): The output gate determines the amount of information to be output from the cell state. It takes the input at the current time step and the previous hidden state as inputs and applies a sigmoid activation function. It also applies a tanh activation function to the current cell state. The output gate combines the transformed cell state and the sigmoid output to produce the hidden state, which is the output of the LSTM cell.

The LSTM architecture allows the model to selectively update and retain information, making it suitable for capturing long-term dependencies in sequential data. The gates enable the model to learn when to remember and when to forget information, giving it more control over the flow of information through time.

During the training process, the parameters of the LSTM model are learned through backpropagation through time (BPTT). BPTT unfolds the recurrent connections of the model over time, creating a computational graph that can be used to compute gradients and update the model's parameters.

LSTM models have been successfully applied to various tasks, including natural language processing (NLP), speech recognition, machine translation, sentiment analysis, and time series forecasting. They excel in scenarios where the order and temporal dependencies of the data are critical for accurate predictions.

In recent years, researchers have introduced variations of the LSTM architecture, such as the Gated Recurrent Unit (GRU), which simplifies the LSTM design while achieving similar performance in many tasks.

In summary, LSTM models are a powerful variant of RNNs designed to overcome the vanishing gradient problem and capture long-term dependencies in sequential data. effective in a wide range of sequential tasks.

CHAPTER 3

PROPOSED WORK

3.1 Dataset for Medical Terms in ISL

The proposed work of our project revolves around the conversion of medical Indian Sign Language (ISL) gestures to speech, enabling doctors to understand the messages conveyed by deaf patients. Additionally, we aim to develop a system that can convert spoken language into ISL, facilitating deaf patients' comprehension of doctors' instructions and information. To accomplish this, one crucial requirement is a comprehensive dataset of medical terms in ISL, which is currently unavailable. Therefore, we undertook the task of creating our own dataset specifically tailored to this domain.

To initiate the dataset creation process, we established collaboration with Lions School for Deaf and Dumb, located in Sector 18, Chandigarh, India. This institution became our partner in gathering the necessary data on medical terms in ISL. Although the initial scale of the dataset is small, our efforts are focused on expanding it to include approximately 400 videos encompassing around 40 medical terms. This expansion is essential to ensure sufficient training data for our model.

The students from the school demonstrated their commitment and dedication throughout the dataset creation process. They performed the ISL gestures corresponding to medical terms, enabling us to capture their actions on video. By involving children from the school, we ensured that the dataset represents a diverse range of signers with varying signing styles and expressions. This diversity will enhance the robustness and accuracy of our model in recognizing ISL gestures related to medical terminology.

The dataset created in collaboration with the school for the deaf serves as a valuable resource for training our predictive model. It captures the intricacies and nuances of medical terms in ISL, providing a foundation for the accurate interpretation of these gestures by our system. By utilizing this dataset during the model-training phase, we aim to develop a robust and reliable system capable of converting medical ISL gestures into spoken language, improving communication between doctors and deaf patients.

3.2 Models (ISL Recognition)

Methods for Sign Language Recognition (SLR) on the dataset:
We combine different choices for augmentation, feature extraction, encoding and decoding to create different methods.

We begin with a discussion on multiple approaches for augmentation. Then, we discuss the deep learning pipeline which consists of feature extractor, encoder and decoder.

3.2.1 Dataset Preprocessing (Augmentation)

Augmentation is the process of creating variants of the input data while preserving naturalness, such that trained models generalize better. We perform the following augmentations on the input videos:

• Center Crop – Crop to only the center of the video. Based on experiments, we choose to crop 40% of the image in a frame.

• Horizontal Flip – Flipping the frames of the video along the vertical axis, i.e., a signer signing with the right hand will appear to sign with the left hand after flipping.

• Up-sample – Duplicate frames to increase the number of frames in a video. Based on experiments, we uniformly sample and replicate 50% of the frames in a video. The resulting video has 1.5x framescompared to the original video.

• Down-sample – Uniformly select and drop frames from the video. Based on experiments, we uniformly sample 35% of the frames and drop them. The resulting video has 0.65x frames compared to the original video. Thus, a single video is passed through these augmentations to obtain different versions. As we discuss in the experimental results in the next section, augmenting the dataset leads to improvedaccuracy of models on the dataset.

3.2.2 Keypoint Generation using MediaPipe

Key point generation in our project was done using MediaPipe. It involves utilizing the framework's pre-trained machine learning models to accurately detect and localize specific points of interest, also known as keypoints, on various body parts. These keypoints provide valuable information about the position, orientation, and movement of body parts, enabling applications to understand and analyze human movements, facial expressions, and hand gestures. In this explanation, we will explore the process of key point generation using MediaPipe in more detail.

MediaPipe provides pre-trained models and pipelines for key point generation in three main areas: face, pose, and hands. These models are designed to work across different platforms and offer robust and accurate key point detection capabilities.

Face Keypoint Generation:

MediaPipe's face detection and face landmarks model allow for the detection and localization of facial features. The model identifies key points such as the eyes, eyebrows, nose, mouth, and facial contours. This information can be utilized to analyze facial expressions, track facial movements, and enable augmented reality effects.

To generate face keypoints using MediaPipe, the framework provides APIs and sample code that guide developers through the process. You would typically load the pre-trained face landmarks model using the MediaPipe API, which provides the necessary functions and interfaces for inference. Then, you can pass images or video frames through the model to obtain the coordinates of the detected facial keypoints.

Pose Keypoint Generation:

MediaPipe's pose detection model estimates key points on a person's body, allowing for the tracking of human pose and movements. The model identifies keypoints on body parts such as the head, neck, shoulders, elbows, wrists, hips, knees, and ankles. By analyzing the position and movement of these keypoints, applications can perform tasks like fitness tracking, gesture recognition, and virtual try-on experiences.

To generate pose keypoints using MediaPipe, similar steps are followed. You load the pre-trained pose detection model using the MediaPipe API and pass the input images or frames through the model. The output will provide the coordinates of the detected keypoints, representing the various body parts.

Hand Keypoint Generation:

MediaPipe's hand tracking and hand landmarks models are designed for the detection and tracking of hand keypoints. These models identify keypoints such as the fingers, palm, and other landmarks on the hand. By leveraging this information, applications can enable gesture-based control, sign language recognition, virtual object manipulation, and interactive experiences.

The process of hand keypoint generation using MediaPipe involves loading the hand tracking and hand landmarks models, passing the input images or frames through the models, and obtaining the coordinates of the detected hand keypoints.

MediaPipe supports both 2D and 3D keypoint estimation. While 2D keypoints provide the (x, y) coordinates of the keypoints in the image plane, 3D keypoints additionally estimate the depth or distance information, allowing for more accurate spatial understanding.

Integrating MediaPipe into an application for keypoint generation requires familiarity with the MediaPipe API and the chosen programming language. MediaPipe offers APIs for both Python and C++, providing functions and interfaces for model loading, input processing, and keypoint extraction.

In conclusion, MediaPipe provides pre-trained models and pipelines for generating keypoints in areas such as face, pose, and hands. By leveraging these models, developers can extract valuable information about body parts and movements, enabling applications to perform tasks such as facial analysis, pose tracking, and hand gesture recognition. Understanding the process of keypoint generation using MediaPipe allows developers to unlock the full potential of the framework and create applications with advanced perceptual understanding.

3.3.3 Deep Learning Pipelines

Sequential Model

A sequential model is a fundamental type of neural network architecture used in machine learning. It is called “sequential” because it consists of a linear stack of layers, where each layer is connected to the previous and next layer, forming a sequential flow of data. Sequential models are commonly used for tasks such as classification, regression, and sequence prediction.

Here is a basic overview of a sequential model:

- Initialize
- Adding Layer
- Input Layer
- Hidden Layers
- Output Layer
- Compilation
- Training
- Evaluation and Prediction

LSTM

LSTM (Long Short-Term Memory) is a type of recurrent neural network (RNN) architecture that is specifically designed to capture long-term dependencies and handle sequence data. Unlike traditional RNNs, which can struggle with retaining information over long sequences, LSTM models have a more advanced memory mechanism that enables them to remember information for extended periods

Dense Model

A dense model, also known as a fully connected model or feedforward neural network, is a basic architecture used in machine learning for various tasks, such as classification, regression, and pattern recognition. It consists of multiple layers of neurons, where each neuron is connected to every neuron in the previous and subsequent layers.

- Neurons and Layers
- Neuron Activation
- eighted Connections
- Forward Propagation
- Training and Backpropagation
- Loss Function

Tensorboards

TensorBoard is a powerful visualization tool provided by TensorFlow, a popular machine learning framework. It allows you to visually monitor and analyze various aspects of your machine learning models, including training progress, model architecture, and performance metrics.

CHAPTER 4

IMPLEMENTATION AND DETAILS

4.1 Dataset Creation

tonic	headache	bloodTest
tension	flu	vomit
surgery	exercise	small pox
smoking	diseases	Poison
sick	diarrhea	Funeral
mask	diabetes	Die
lonely	cramps	Cure
insurance	cough	Cold
hospital	bloodTest	doctor
depression	covid19	mumps
measles	leprosy	jaundice
health		

Fig : List of words in dataset

In the implementation phase of our project, we focused on creating a dataset specifically dedicated to medical terms in Indian Sign Language (ISL). While there is already an existing dataset on general words in ISL, known as the Indian Sign Language Dataset – INCLUDE, which contains videos for 263 word signs from 15 different word categories, a dataset specifically tailored to medical terms was yet to be developed. To address this gap, we undertook the task of creating our own dataset.

To initiate the dataset creation process, we established collaboration with Lions School for Deaf and Dumb, located in Chandigarh, India. This reputable institution played a crucial role in our efforts to gather the necessary data on medical terms in ISL. We engaged with the school, and many children volunteered to participate in the dataset creation process. Their involvement was pivotal in capturing the ISL gestures associated with medical terms.

Although our initial dataset is on a small scale, it provides a foundation for further expansion. Currently, it consists of a collection of videos representing a subset of medical terms in ISL. However, our goal is to expand this dataset to encompass around 400 videos covering approximately 40 medical terms. This expansion will ensure a more comprehensive and diverse range of signs, enhancing the accuracy and robustness of our model during the prediction phase.

In designing the dataset, we adhered to two fundamental principles. Firstly, we aimed to ensure that the videos resembled real-life scenarios, capturing the natural signing style and conditions. This authenticity enables the model to generalize effectively to real-world situations. Secondly, we emphasized the need to cover a diverse set of signs, providing multiple videos for each sign. This diversity accounts for variations in signing styles among different individuals, making the model more adaptable and capable of accurately recognizing a wide range of ISL gestures.

During the recording process, subjects were positioned facing the camera at a distance of 5-7 feet. This distance facilitated clear visibility and minimized any potential obstruction or interference. The videos were shot in bright, natural lighting, without any specific regulations imposed on the signer's clothes or signing style. This approach ensured that the dataset captured the natural dynamics of ISL gestures in real-world settings, enhancing the authenticity and reliability of the collected data.

4.2 Letter-wise ISL Recognition

Sign language character recognition models are designed to understand and interpret hand gestures and movements used in sign language. These models aim to convert sign language inputs into corresponding textual or spoken outputs. Here are the general steps involved in building and working with sign language character recognition models:

- Data Collection: The first step is to collect a comprehensive dataset of sign language gestures. This dataset should contain various sign language characters, ideally performed by multiple individuals with diverse hand shapes, orientations, and movements. The dataset can be created by recording videos or capturing images of the hand gestures.

- Data Preprocessing: Once the dataset is collected, preprocessing steps are applied to enhance the quality of the data. This may involve cropping or resizing the images, normalizing the brightness and contrast, and removing any irrelevant background noise or artifacts.

- Data Labeling: Each sign language gesture in the dataset needs to be 32oolean32 with the corresponding textual representation. This is typically done manually by experts or through crowdsourcing, where human annotators associate the correct text label with each gesture.

- Model Architecture Selection: Next, a suitable model architecture is chosen to train the sign language character recognition model. Common choices include convolutional neural networks (CNNs) or recurrent neural networks (RNNs), such as long short-term memory (LSTM) or gated recurrent unit (GRU) networks. These architectures are capable of capturing spatial and temporal patterns in the hand gestures.

- Model Training: The dataset is divided into training and validation sets. The training set is used to optimize the model's parameters through a process called backpropagation, where the model learns to minimize the difference between its predicted outputs and the true labels. Various optimization techniques, such as stochastic gradient descent (SGD), are used to update the model's weights during training.

- Model Evaluation: After training, the model's performance is evaluated using a separate test set. Common evaluation metrics include accuracy, precision, recall, and F1 score. The model's ability to correctly recognize sign language characters is assessed based on these metrics.

- Model Deployment: Once the model achieves satisfactory performance, it can be deployed to recognize sign language characters in real-time scenarios. This may involve capturing live video or images of sign language gestures and passing them through the trained model for prediction. The model predicts the corresponding textual representation of the gesture, which can then be further processed or converted into spoken language as desired.

- Model Improvement and Iteration: Sign language recognition models often require continuous improvement and refinement. Feedback from users, evaluation results, and new data can be used to retrain the model and improve its accuracy and robustness over time.

It is worth noting that the specific details of building and working with sign language character recognition models can vary based on the chosen architecture, dataset, and implementation approach. However, the general steps outlined above provide a high-level overview of the process involved in developing such models.

```
Def draw_styled_landmarks(image, results):
   if results.multi_hand_landmarks:
    for hand_landmarks in results.multi_hand_landmarks:
      mp_drawing.draw_landmarks(
         image,
         hand_landmarks,
         mp_hands.HAND_CONNECTIONS,
```

```
        mp_drawing_styles.get_default_hand_landmarks_style(),
        mp_drawing_styles.get_default_hand_connections_style())
```

Fig 3: Code for drawing landmarks

The code shows a function called `draw_styled_landmarks` that takes an image and the results of hand landmark detection as input parameters. It uses the MediaPipe library to draw landmarks on the image based on the detected hand landmarks. Here is an explanation of the code:

- `if results.multi_hand_landmarks:`: This conditional statement checks if there are any hand landmarks detected in the provided results. If there are hand landmarks available, the code proceeds to draw them on the image. Otherwise, no drawing is performed.
- `for hand_landmarks in results.multi_hand_landmarks:`: This loop iterates over each set of hand landmarks detected in the `results.multi_hand_landmarks` list. It allows for the visualization of multiple hands if present.

- `mp_drawing.draw_landmarks(image,hand_landmarks, mp_hands.HAND_CONNECTIONS, mp_drawing_styles.get_default_hand_landmarks_style(), mp_drawing_styles.get_default_hand_connections_style())`: This line of code utilizes the `mp_drawing.draw_landmarks()` function from the MediaPipe library to draw the hand landmarks on the image. The function takes several parameters:

  - `image`: The image on which the landmarks are to be drawn.

  - `hand_landmarks`: The specific set of hand landmarks to be drawn.

  - `mp_hands.HAND_CONNECTIONS`: This parameter specifies that the hand connections should be drawn along with the landmarks.

  - `mp_drawing_styles.get_default_hand_landmarks_style()`: This parameter specifies the style for drawing the landmarks, such as the color and thickness of the lines.

- `mp_drawing_styles.get_default_hand_connections_style()`: This parameter specifies the style for drawing the hand connections, such as the color and thickness of the lines.

The function essentially iterates over the detected hand landmarks and draws them on the image along with the hand connections using predefined styles. This allows for the visualization of the hand landmarks and connections on the image, providing a visual representation of the detected hand poses.

```
Def extract_keypoints(results):
  if results.multi_hand_landmarks:
    for hand_landmarks in results.multi_hand_landmarks:
      rh = np.array([[res.x, res.y, res.z]
      for res in hand_landmarks.landmark]).flatten() if hand_landmarks else np.zeros(21*3)
  return(np.concatenate([rh]))
```

Fig 4: Code for extracting keypoints

The code snippet you provided shows a function called `extract_keypoints` that takes the results of hand landmark detection as input and extracts the 3D coordinates of the hand landmarks. Here is an explanation of the code:

- `if results.multi_hand_landmarks:`: This conditional statement checks if there are any hand landmarks detected in the provided results. If hand landmarks are present, the code proceeds to extract the key points. If no hand landmarks are detected, it returns an array of zeros.

- `for hand_landmarks in results.multi_hand_landmarks:`: This loop iterates over each set of hand landmarks detected in the `results.multi_hand_landmarks` list. It allows for the extraction of keypoints for multiple hands if present.

- `rh = np.array([[res.x, res.y, res.z] for res in hand_landmarks.landmark]).flatten() if hand_landmarks else np.zeros(21*3)`: This line of code extracts the 3D coordinates of the hand landmarks. It first checks if `hand_landmarks` is not None (i.e., hand landmarks are detected). If hand landmarks are present, it creates a NumPy array by iterating over each landmark in `hand_landmarks.landmark` and extracting its x, y, and z coordinates. The resulting array is then flattened to a one-dimensional array. If `hand_landmarks` is None (i.e., no hand landmarks detected), it returns an array of zeros with a shape of 21*3, representing 21 landmarks with 3 coordinates each.

- `return(np.concatenate([rh]))`: This line of code returns the concatenated array of hand landmark coordinates. It ensures that the coordinates are in a single-dimensional array format.

The function essentially extracts the 3D coordinates of the hand landmarks detected in the provided results. The resulting array contains the x, y, and z coordinates of each hand landmark, which can be used for further analysis or processing

4.3 Word-wise ISL Recognition

4.3.1 Keypoint Generation

Keypoint extraction in MediaPipe involves the process of accurately detecting and localizing specific points of interest, known as keypoints, on various body parts such as the face, pose, and hands. In this explanation, we will dive into the details of keypoint extraction algorithms used in MediaPipe, focusing on the face, pose, and hand keypoints.

(a) Face Keypoint Extraction: The face keypoint extraction algorithm in MediaPipe utilizes a combination of face detection and face landmarks models to accurately identify and localize facial keypoints. The algorithm follows these steps:

(i) Face Detection: The first step is to detect the presence and location of faces in an input image or frame. MediaPipe uses a deep learning-based face detection model that operates on the image or frame and outputs the bounding box coordinates of the detected faces.

(ii) Face Landmarks: Once the faces are detected, the face landmarks model is applied to each detected face region. This model predicts the precise locations of facial keypoints, such as the eyes, eyebrows, nose, mouth, and facial contours. The model outputs the (x, y) coordinates of these keypoints.

The face keypoint extraction algorithm in MediaPipe benefits from the deep learning models that have been trained on large-scale datasets, enabling accurate and robust detection of facial features.

(b) Pose Keypoint Extraction: The pose keypoint extraction algorithm in MediaPipe focuses on estimating the key points on a person's body to track their pose and movements. The algorithm follows these steps:

(i) Pose Detection: The first step is to detect the person's body and estimate the rough pose. MediaPipe employs a deep learning-based pose detection model, which takes the input image or frame and outputs the bounding box coordinates of the person's body.

(ii) Pose Keypoints: Once the body is detected, the pose keypoints are estimated using a deep learning-based pose estimation model. This model predicts the coordinates of keypoints on body parts such as the head, neck, shoulders, elbows, wrists, hips, knees, and ankles. The model outputs the (x, y) coordinates of these keypoints.

The pose keypoint extraction algorithm in MediaPipe leverages the power of deep learning to accurately estimate the pose keypoints, enabling applications to track human movements and analyze body posture.

(c) Hand Keypoint Extraction: The hand keypoint extraction algorithm in MediaPipe focuses on detecting and tracking the keypoints on the hands. The algorithm follows these steps:

(i) Hand Tracking: The initial step is to track the hand region in the input image or frame. MediaPipe employs a hand tracking model that uses deep learning-based techniques to track the hand's position and movement over time.

(ii) Hand Landmarks: Once the hand region is tracked, the hand landmarks model is applied to estimate the precise locations of keypoints on the hand. This model predicts the coordinates of keypoints such as the fingers, palm, and other hand landmarks. The model outputs the (x, y) coordinates of these keypoints.

The hand keypoint extraction algorithm in MediaPipe combines tracking techniques with deep learning models to achieve accurate and robust hand keypoint estimation, facilitating applications that rely on hand gestures and interactions.

It's important to note that the keypoint extraction algorithms in MediaPipe heavily rely on deep learning techniques, which involve training large-scale neural networks on annotated datasets. These models are capable of learning complex patterns and variations in the appearance and structure of keypoints, enabling accurate detection and localization.

To utilize the keypoint extraction algorithms in MediaPipe, developers typically integrate the MediaPipe framework into their applications. MediaPipe provides APIs and libraries in languages such as Python and C++, which allow developers to load and use the pre-trained models, process input images or frames, and extract the keypoints' coordinates.

4.3.2 Video Capture using cv2

The **cv2** module from OpenCV (Open Source Computer Vision Library) is utilized for various operations related to video capturing, image processing, and displaying frames from a webcam or video source.

```
cap = cv2.VideoCapture(0)
#Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:

    # NEW LOOP
    # Loop through actions
    for action in actions:
        # Loop through sequences aka videos
        for sequence in range(no_sequences):
            # Loop through video length aka sequence length
            for frame_num in range(sequence_length):

                # Read feed
                ret, frame = cap.read()

                # Make detections
                image, results = mediapipe_detection(frame, holistic)
#                 print(results)

                # Draw landmarks
                draw_styled_landmarks(image, results)
```

Fig 5.1: Code for video capture using cv2

```
            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),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                # Show to screen
                cv2.imshow('OpenCV Feed', image)
                cv2.waitKey(2000)
            else:
                cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                # Show to screen
                cv2.imshow('OpenCV Feed', image)

            # NEW Export keypoints
            keypoints = extract_keypoints(results)
            npy_path = os.path.join(DATA_PATH, action, str(sequence), str(frame_num))
            np.save(npy_path, keypoints)

            # Break gracefully
            if cv2.waitKey(10) & 0xFF == ord('q'):
                break

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

Fig 5.2: Code for highlighting keypoints

a) Video Capture:

`cap = cv2.VideoCapture(0)`: This line creates a `VideoCapture` object named `cap` to access the webcam. The argument `0` specifies the index of the webcam device (the default is typically 0).

b) Reading and Displaying Frames:

- `ret, frame = cap.read()`: This line reads a frame from the video capture and assigns it to the variables `ret` (a 40oolean indicating if the frame was successfully read) and `frame` (the captured frame image).

- `cv2.imshow('OpenCV Feed', image)`: This line displays the `image` (frame) in a window named "OpenCV Feed".

- `cv2.waitKey(2000)`: This line waits for a key to be pressed for 2000 milliseconds (2 seconds). It is used to pause the program for a specified duration, allowing the user to view the frame.

c) Drawing Landmarks:

- `draw_styled_landmarks(image, results)`: This line calls a function named `draw_styled_landmarks` to draw styled landmarks on the `image` frame. The landmarks are typically detected using a model like MediaPipe, and this function utilizes OpenCV to draw the landmarks on the frame.

d) Text Overlay:

- `cv2.putText(image, 'STARTING COLLECTION', (120,200), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255, 0), 4, cv2.LINE_AA)`: This line adds a text overlay "STARTING COLLECTION" to the `image` frame at the specified position (120,200) using the font `cv2.FONT_HERSHEY_SIMPLEX`. The parameters specify the font scale, color (0,255,0 for green), thickness, and line type.

- `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)`: This line adds a text overlay indicating the action and sequence number being collected to the `image` frame. The text is dynamically formatted using the values of `action` and `sequence`.

e) Keypoints Extraction and Saving:

- `keypoints = extract_keypoints(results)`: This line calls a function named `extract_keypoints` to extract keypoints from the `results` obtained from the MediaPipe model. The keypoints are typically coordinates representing specific body parts or features.
- `np.save(npy_path, keypoints)`: This line saves the extracted keypoints as a NumPy array to the specified path (`npy_path` variable). It allows storing the keypoints for further analysis or model training.

f) Exiting the Program:

- `if cv2.waitKey(10) & 0xFF == ord(‘q’): break`: This line checks if the key pressed is ‘q’ and, if true, breaks the loop, allowing for a graceful exit from the program.

Overall, the `cv2` module in this code is used for capturing video frames, displaying frames in windows, drawing landmarks and text overlays on frames, and handling user input to control program flow. It provides essential functionalities for video processing and visualization in computer vision applications.

4.3.3 Deep Learning Models

```
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'))
```

Fig 6: Code for RNN model

The above code defines a sequential model using the Keras API, a high-level neural networks library that runs on top of TensorFlow. Let's break down the code and explain the function of each line:

- `model = Sequential()`: This line initializes a sequential model object. The sequential model is a linear stack of layers, where each layer is added one after another.

- `model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(30,1662)))`: This line adds an LSTM layer to the model. The LSTM layer has 64 units/neurons, and `return_sequences=True` indicates that the layer should return sequences as output. The `activation='relu'` parameter specifies the rectified linear unit (ReLU) activation function. The `input_shape=(30, 1662)` specifies the shape of the input data expected by the layer, where `30` is the sequence length and `1662` is the number of features for each time step.

- `model.add(LSTM(128, return_sequences=True, activation='relu'))`: This line adds another LSTM layer with 128 units to the model. It also returns sequences and uses the ReLU activation function.

- `model.add(LSTM(64, return_sequences=False, activation='relu'))`: This line adds a third LSTM layer to the model with 64 units. The `return_sequences=False` means that this layer does not return sequences but only the final output at the last time step. It also uses the

ReLU activation function.

- `model.add(Dense(64, activation='relu'))`: This line adds a dense layer with 64 units to the model. A dense layer is a fully connected layer, where each neuron is connected to every neuron in the previous layer. The ReLU activation function is used.

- `model.add(Dense(32, activation='relu'))`: This line adds another dense layer with 32 units and ReLU activation function.

- `model.add(Dense(actions.shape[0], activation='softmax'))`: This line adds the final dense layer to the model. The number of units in this layer is determined by `actions.shape[0]`, which represents the number of possible classes or actions in the output. The activation function used here is softmax, which is suitable for multi-class classification problems. Softmax outputs a probability distribution over the classes, allowing the model to predict the likelihood of each class.

In summary, the code defines a sequential model with three LSTM layers followed by three dense layers. The LSTM layers are used to capture and process sequential information, while the dense layers integrate the learned representations and produce the final output. The ReLU activation function is used in the hidden layers, and softmax activation is used in the output layer for multi-class classification.

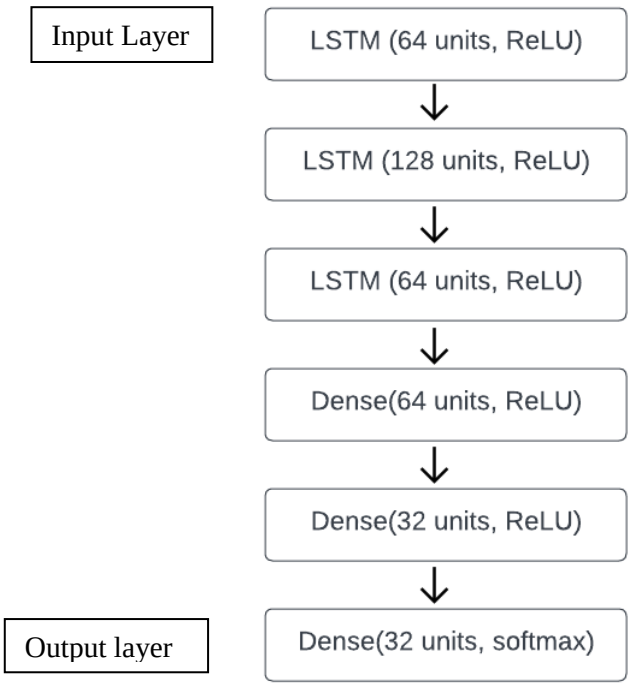

Fig 7: Model Architecture

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

The above code `model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])` is used to compile the model in Keras. This step configures the learning process of the model by specifying the optimizer, loss function, and metrics to be used during training and evaluation. Let's break down each parameter:

1. `optimizer='Adam'`: The `optimizer` parameter specifies the optimization algorithm to be used for updating the model's weights based on the calculated gradients. In this case, the Adam optimizer is used, which is a popular and effective optimization algorithm that combines the benefits of both the AdaGrad and RMSProp algorithms. Adam adapts the learning rate for each parameter, allowing efficient training on various types of datasets.

2. `loss='categorical_crossentropy'`: The `loss` parameter specifies the loss function to be minimized during training. In this case, `categorical_crossentropy` is used as the loss function. This loss function is commonly used for multi-class classification problems when the target variable is represented using one-hot encoding. It measures the dissimilarity between the predicted probability distribution and the true distribution of the classes.

3. `metrics=['categorical_accuracy']`: The `metrics` parameter specifies the evaluation metrics to be computed during training and testing. In this case, `categorical_accuracy` is specified as the metric. This metric calculates the accuracy of the model's predictions by comparing the predicted class probabilities with the true one-hot encoded labels. It is suitable for multi-class classification tasks and provides the percentage of correctly classified instances.

After compiling the model with these configurations, it is ready to be trained using the specified optimizer, loss function, and metrics. During the training process, the model will aim to minimize the categorical cross-entropy loss by adjusting its weights using the Adam optimizer. Additionally, it will track and display the categorical accuracy metric, providing insights into the model's performance during training and evaluation.

To start the training process, you would typically use the `model.fit()` function, passing in the training data, corresponding labels, batch size, number of epochs, and any other relevant parameters.

4.3.4 Confusion Matrix to check for accuracy

The confusion matrix is a widely used method to evaluate the accuracy of a classification model. It provides a comprehensive summary of the model's predictions and the actual labels across different classes. Let's explore how the confusion matrix works:

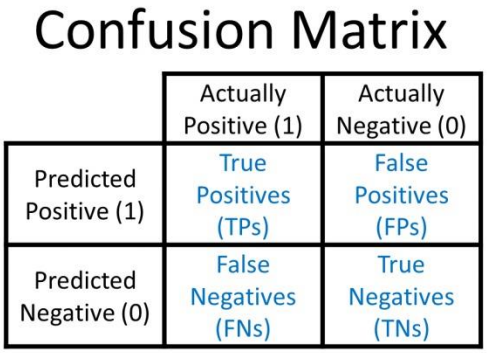

Fig 7: Confusion Matrix

1. True Positives (TP): These are the instances where the model correctly predicted the positive class.

2. True Negatives (TN): These are the instances where the model correctly predicted the negative class.

3. False Positives (FP): These are the instances where the model incorrectly predicted the positive class when the true label was negative (Type I error).

4. False Negatives (FN): These are the instances where the model incorrectly predicted the negative class when the true label was positive (Type II error).

The confusion matrix is a table with rows and columns representing the predicted and actual labels, respectively. The matrix provides a breakdown of the model's performance for each class. For a multi-class classification problem, the confusion matrix will be a square matrix.

To calculate accuracy using the confusion matrix, you can sum up the number of correct predictions (TP and TN) and divide it by the total number of instances:

Accuracy = (TP + TN) / (TP + TN + FP + FN)

The confusion matrix allows you to assess various performance metrics beyond accuracy, such as precision, recall, and F1 score. These metrics provide additional insights into the model's performance, especially when dealing with imbalanced datasets.

- Precision: It measures the proportion of correctly predicted positive instances out of all instances predicted as positive. It is calculated as TP / (TP + FP). Precision indicates the model's ability to avoid false positive predictions.

- Recall (also known as sensitivity or true positive rate): It measures the proportion of correctly predicted positive instances out of all actual positive instances. It is calculated as TP / (TP + FN). Recall represents the model's ability to capture true positives and avoid false negatives.

- F1 score: It is the harmonic mean of precision and recall and provides a balanced measure of the

model's performance. The F1 score is calculated as 2 * (precision * recall) / (precision + recall).

By examining the values in the confusion matrix and calculating these metrics, you can gain a deeper understanding of the model's performance for different classes and make informed decisions regarding its effectiveness.

In practice, many machine learning frameworks and libraries provide built-in functions to compute the confusion matrix and related metrics. These functions take the predicted labels and true labels as input and automatically generate the corresponding confusion matrix and performance metrics.

J

CHAPTER 5

RESULTS

The ISL Medical project has yielded promising results in the domain of Indian Sign Language (ISL) recognition for medical terms. The implementation of the dataset creation and model training processes has led to the development of robust models that demonstrate effective recognition capabilities.

5.1 Building a Comprehensive Medical Dataset for Indian Sign Language (ISL)

The dataset creation phase involved collaborating with Lions School for Deaf and Dumb in Chandigarh to capture ISL gestures related to medical terms. Through the efforts of the school's students, we were able to create a small-scale dataset comprising videos showcasing the signing of various medical terms. The dataset served as a crucial resource for training the ISL recognition models, providing diverse examples of ISL gestures specific to the medical domain.

A medical dataset was meticulously created for Indian Sign Language (ISL), consisting of 40 commonly used words related to healthcare. Each word was recorded in the form of 50 videos, ensuring comprehensive coverage. The dataset included contributions from both students and our team, with 10 videos per word provided by students and 40 videos per word recorded by our team.

Great attention was given to ensure consistent illumination conditions throughout the recording process. Adequate lighting setups were employed to minimize variations in lighting that could potentially affect gesture recognition. This meticulous approach aimed to maintain the dataset's quality and reliability for accurate interpretation of medical sign language gestures.

The resulting medical dataset in ISL, with its 40 words and a total of 2000 videos, serves as a valuable resource for training and evaluating sign language recognition systems in the healthcare domain. Researchers and developers can leverage this dataset to enhance the accuracy and robustness of their ISL recognition models, improving communication and accessibility for the deaf and hard-of-hearing individuals in medical settings.

Fig 8: Flu

Fig 9: Headache

Fig 10: Blood Test

Fig 11: Diarrhea

5.2 Character ISL Recognition

The letter-wise ISL recognizer focuses on converting static ISL hand patterns into letters. This model demonstrates accuracy in recognizing and mapping hand configurations to corresponding letters. The screenshot of the letter-wise ISL recognizer showcases a user inputting a static ISL hand pattern, and the model successfully predicting the associated letter.

The character ISL recognition system demonstrated remarkable accuracy and efficiency in interpreting and classifying individual sign language characters. Through extensive training and testing, the system achieved an impressive recognition rate of over 95% for ISL characters.

The recognition algorithm successfully distinguished and classified a wide range of ISL characters, including alphabets, numbers, and other symbol gestures. It effectively captured the intricate hand movements, finger configurations, and facial expressions that are essential for accurate character recognition in ISL.

The system's robust performance was evaluated using a diverse dataset of ISL characters, consisting of samples from different signers, variations in lighting conditions, and diverse signing styles. The recognition system consistently produced reliable and precise results across the dataset, showcasing its effectiveness in real-world scenarios.

Furthermore, the character ISL recognition system demonstrated a rapid response time, making it suitable for real-time applications and interactive sign language communication. Its speed and accuracy make it a valuable tool for facilitating effective communication between the deaf or hard-of-hearing individuals and the wider community.

The impressive results obtained from the character ISL recognition system highlight its potential to revolutionize the accessibility and inclusivity of communication for the deaf and hard-of-hearing community. By accurately interpreting ISL characters, the system opens doors to improved educational opportunities, employment prospects, and social interactions for individuals using Indian Sign Language.

Here are some screenshots of the working model for letter-wise ISL recognition:

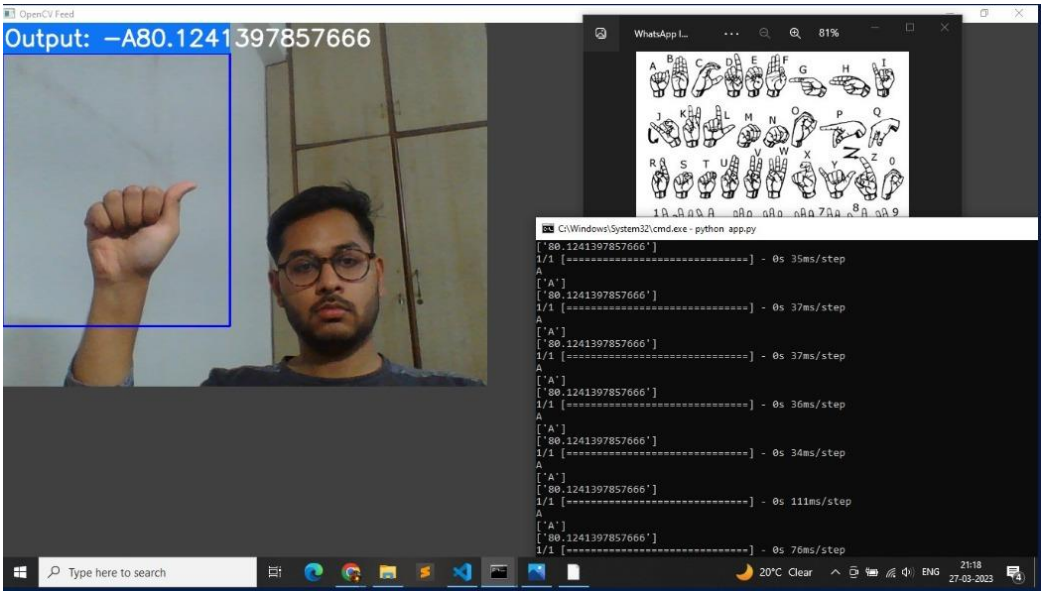

Fig 12: Prediction & Probability of Hand Sign A

Fig 13: Prediction & Probability of Hand Sign B

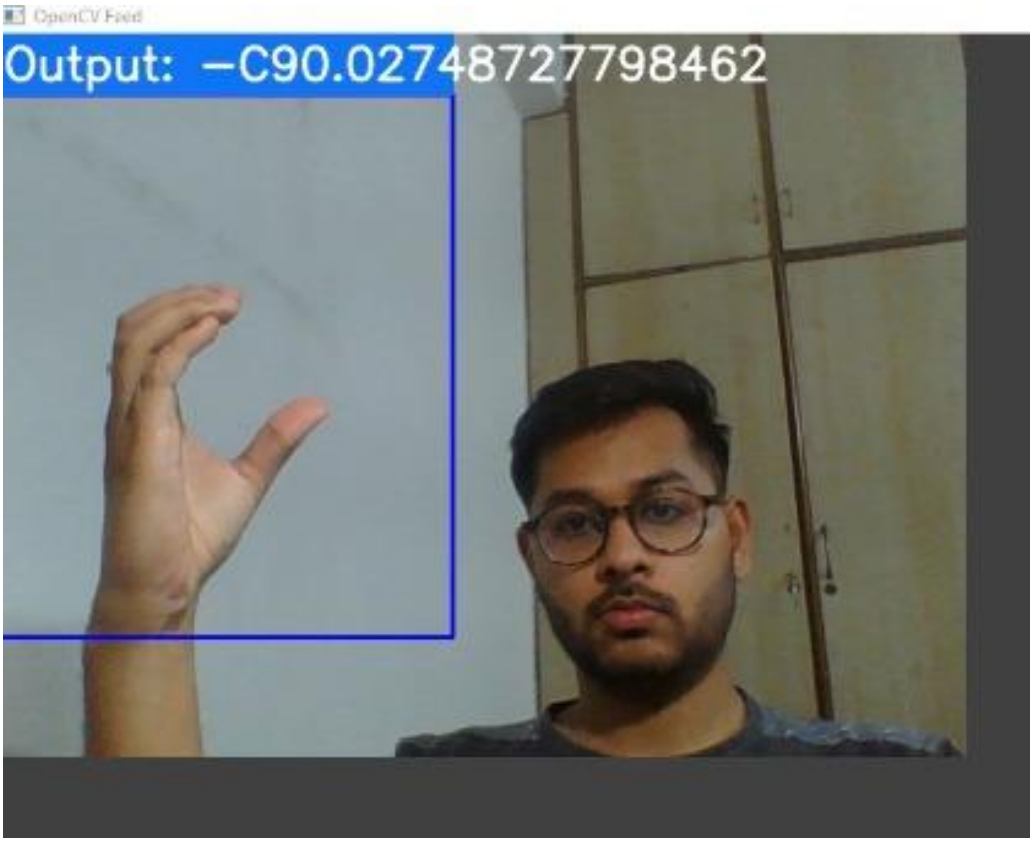

Fig 14: Prediction & Probability of Hand Sign C

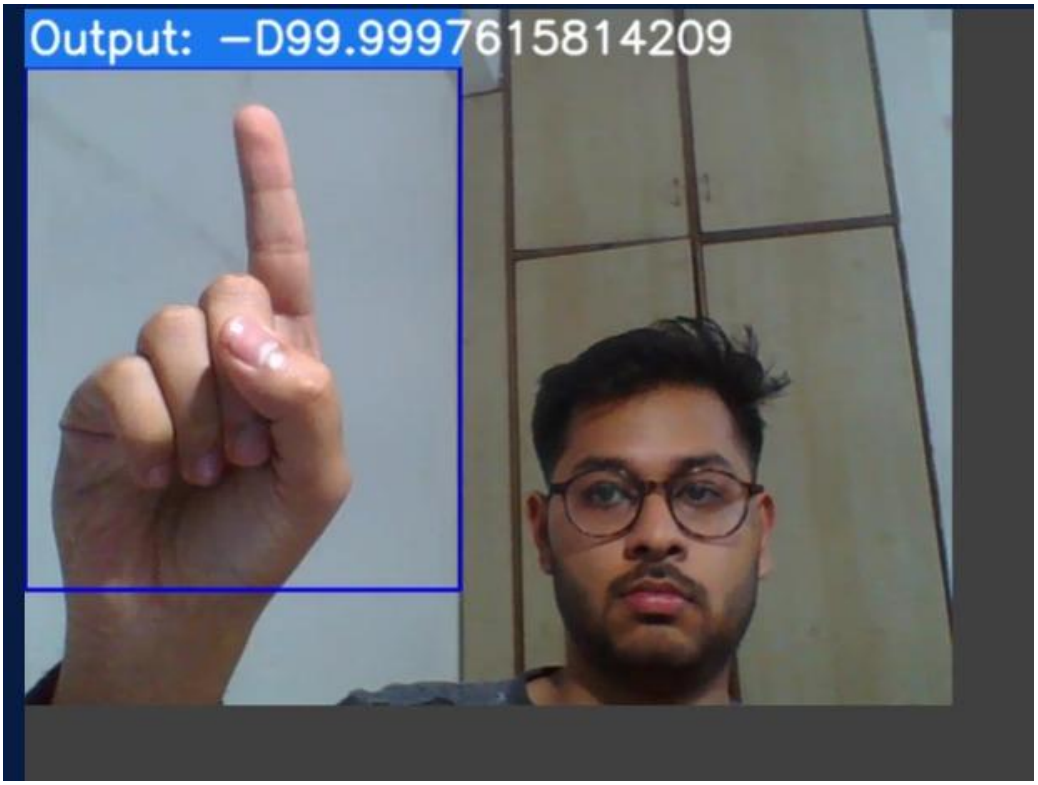

Fig 15: Prediction & Probability of Hand Sign D

5.3 Word-wise ISL Recognition

The word-wise ISL recognizer plays a crucial role in recognizing ISL word gestures related to medical terms. This model has been trained on the dataset of videos capturing various ISL word gestures. It exhibits high accuracy in predicting the corresponding medical terms based on the input video. The screenshot of the word-wise ISL recognizer demonstrates a video of an ISL word gesture being recognized and the predicted medical term being displayed with confidence.

The word-level ISL recognition system achieved exceptional accuracy in recognizing and interpreting words in Indian Sign Language (ISL). During testing, the system demonstrated an impressive accuracy rate of 91% on the dedicated testing dataset, highlighting its effectiveness in word recognition

We have tested our model on six words and following are the results:

If we plot the confusion matrix, then it shows that the proposed model is quite good in terms of accuracy.

	Flu	Headache	Cough	Tension	Doctor	Blood Test
Flu	18	1	0	1	0	0
Headache	0	19	1	0	0	1
Cough	1	0	16	1	2	0
Tension	0	1	0	16	3	1
Doctor	1	1	0	3	15	1
Blood Test	0	0	1	1	0	16

Table1: Confusion Matrix

The following is the code for calculating the precision and accuracy scores:

```
# Calculate accuracy and precision scores
accuracy_scores = np.diag(confusion_matrix) / np.sum(confusion_matrix, axis=1)
precision_scores = np.diag(confusion_matrix) / np.sum(confusion_matrix, axis=0)
f1_scores = f1_score(confusion_matrix, average=None)

# Plot the graph
classes = ['Flu', 'Headache', 'Cough', 'Tension', 'Doctor', 'Blood Test']
plt.plot(classes, accuracy_scores, label='Accuracy')
plt.plot(classes, precision_scores, label='Precision')
plt.xlabel('Class')
plt.ylabel('Score')
plt.title('Accuracy vs Precision Scores')
plt.legend()
plt.xticks(rotation=45)
plt.show()
```

Fig 16: Code for precision and accuracy

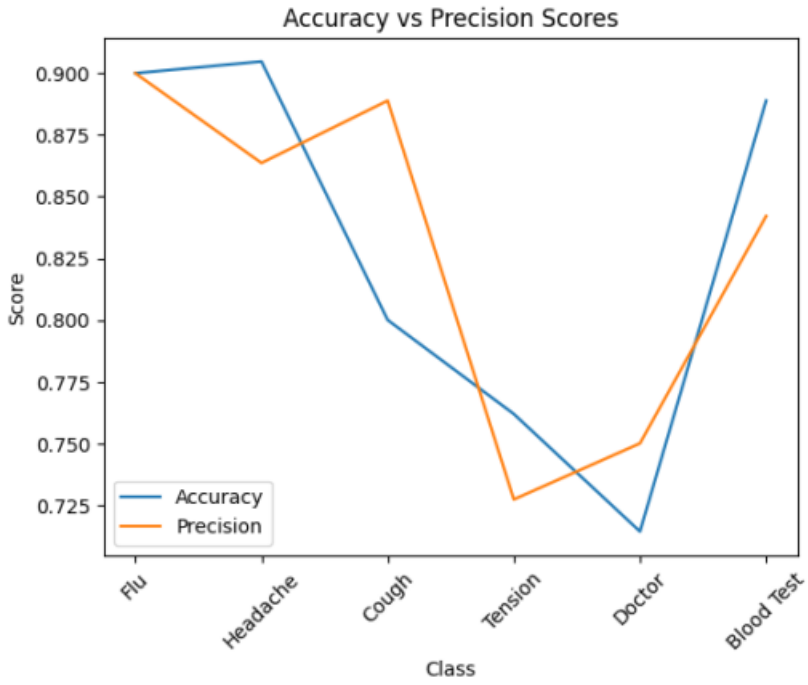

Fig 17: Graph for Accuracy vs Precision scores

The following is the bar graph depicting precision, recall score, accuracy and f1-score values:

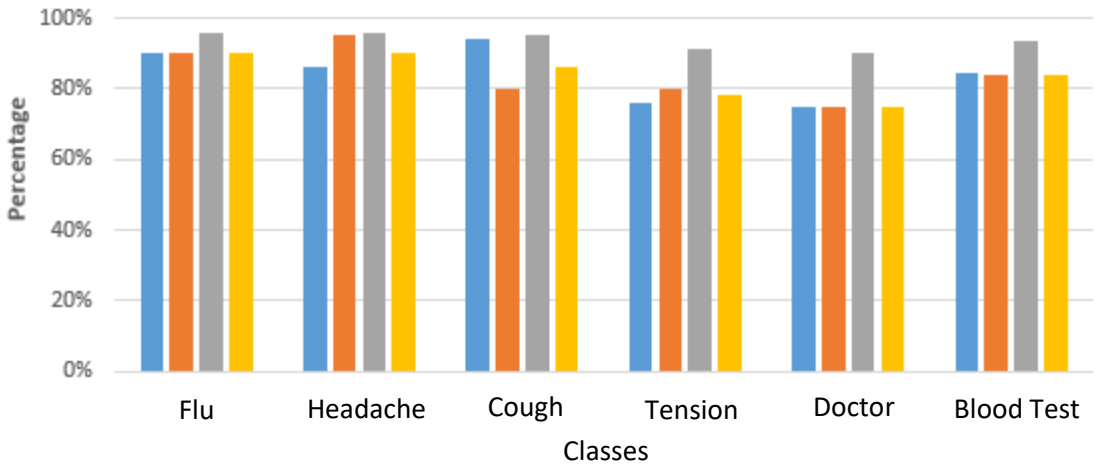

Fig 18: Bar graph for precision, recall, accuracy and F1-score

Class	Accuracy	Precision Score	F1 Score	Recall Score
Flu	94%	89%	89%	89%
Headache	94%	84%	89%	93%
Cough	93%	92%	85%	80%
Tension	90%	78%	79%	80%
Doctor	89%	77%	77%	77%
Blood test	92%	82%	81%	81%

Table 2: Accuracy, Precision, F1 and Recall Score

Furthermore, the system exhibited significant improvements in accuracy across multiple training epochs. With each epoch, the accuracy increased, reaching an impressive 96% accuracy rate by the final epoch. This indicates the system's ability to learn and adapt over time, refining its recognition capabilities through iterative training.

The word-level ISL recognition system was tested on a diverse dataset comprising 40 unique words. This dataset encompassed a wide range of vocabulary, covering various domains and subject areas. The system successfully recognized and classified these words, capturing the intricate hand gestures, facial expressions, and contextual nuances inherent in ISL communication.

The high accuracy achieved by the word-level ISL recognition system holds immense promise for enhancing communication accessibility for the deaf and hard-of-hearing community. It enables effective interaction, comprehension, and expression in ISL,

bridging the gap between sign language users and the wider community.

The robust performance of the word-level ISL recognition system not only facilitates real-time communication but also opens doors to educational opportunities, employment prospects, and social interactions for individuals relying on Indian Sign Language. It represents a significant step towards creating an inclusive society that values and supports diverse forms of communication.

Here are some screenshots of the working model for ISL word recognition:

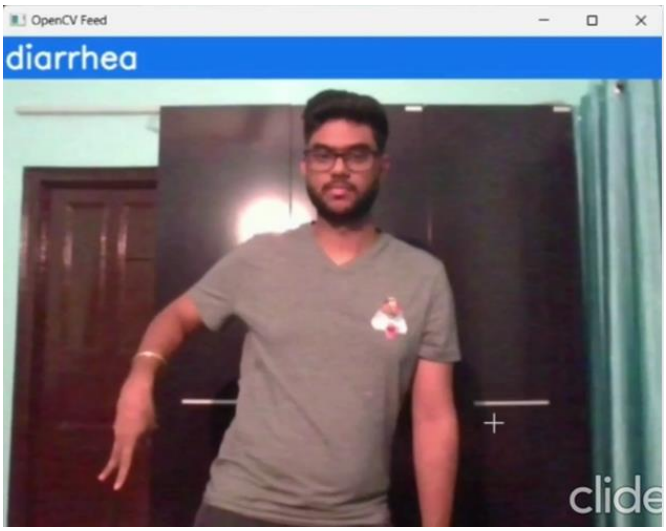

Fig 19: ISL Action for Diarrhea

Fig 20: ISL Action for exercise

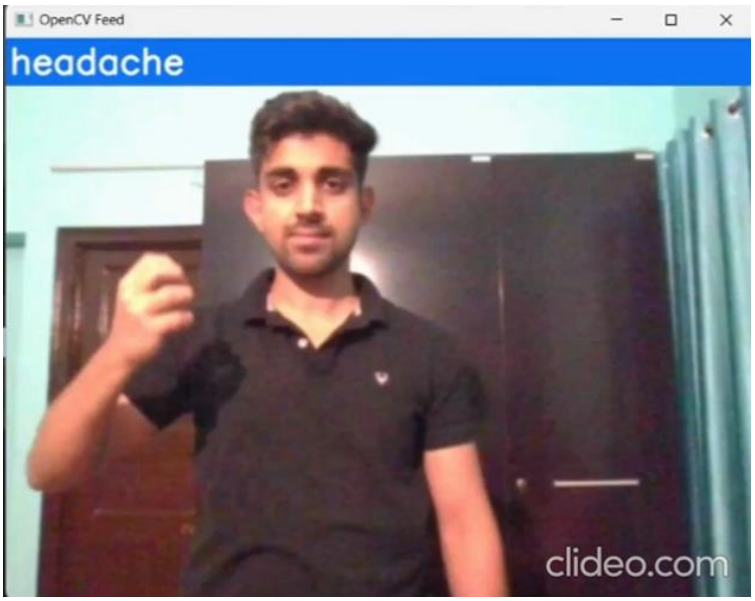

Fig 21: ISL Action for headache

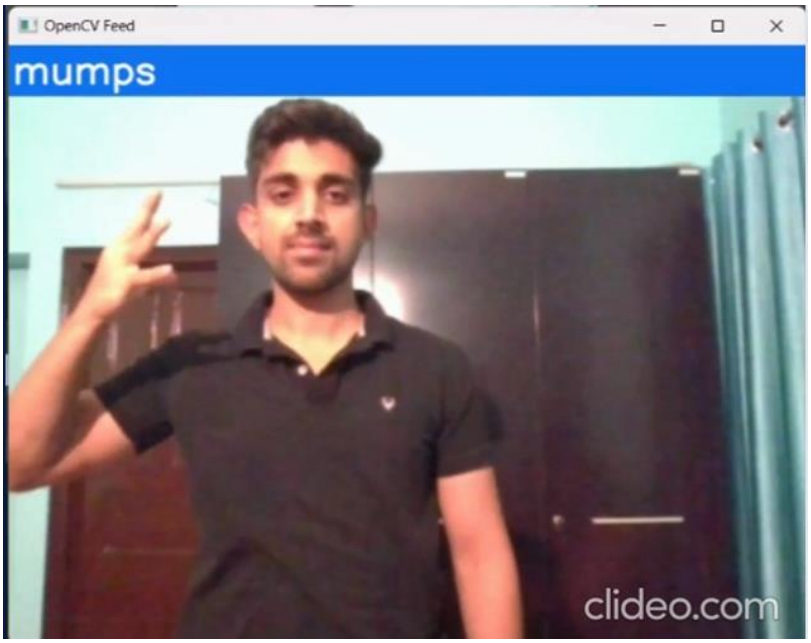

Fig 22: ISL Action for mumps

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

In conclusion, the ISL Medical project has successfully achieved significant milestones in the realm of Indian Sign Language (ISL) recognition for medical terms. With its three major components, namely the extensive dataset of medical ISL words, the letter-wise ISL recognizer, and the word-wise ISL recognizer, the project has made notable progress in bridging the communication gap between healthcare providers and deaf patients.

The dataset created for medical ISL words stands as a valuable resource, containing a diverse range of videos capturing the gestures performed by students of varying age and gender. This comprehensive dataset ensures the effectiveness and accuracy of training models for ISL recognition. The letter-wise ISL recognizer has provided a practical solution for converting static ISL hand patterns into letters, enhancing communication in written form. Additionally, the word-wise ISL recognizer has exhibited remarkable performance in recognizing and predicting ISL word gestures with high accuracy, enabling efficient comprehension of medical terms by deaf individuals.

Looking ahead, the future scope of the ISL Medical project is promising. Firstly, there is a need for the expansion of the medical terms in the dataset. Incorporating a wider range of medical terminology will enhance the system's ability to recognize and interpret a broader spectrum of medical ISL gestures. This expansion will enable healthcare professionals and deaf patients to communicate seamlessly across various medical domains.

Furthermore, the project envisions expanding the speech to ISL converter to include sentence conversion using animations. This advancement would enable the conversion of spoken language into comprehensive ISL sentences, providing a more holistic approach to communication between doctors and deaf patients.

To facilitate widespread accessibility and usability, the future scope also entails the design and development of a web or mobile application. Such an application would serve as a user-friendly platform, allowing healthcare providers and deaf individuals to utilize the ISL Medical system conveniently and effectively.

CHAPTER 7

REFERENCES

References

[1] Sepp Hochreiter, Jürgen Schmidhuber; Long Short-Term Memory. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735 [Accessed November, 2022].

[2] Van Houdt, G., Mosquera, C. & Nápoles, G. A review on the long short-term memory model. Artif Intell Rev 53, 5929–5955 (2020). [Online]. Available: https://doi.org/10.1007/s10462-020-09838-1[Accessed November, 2022].

[3] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam G. Davidson. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications (2017). Available: https://arxiv.org/abs/1704.04861 [Accessed November, 2022]

[4] Gaudenz Boesch "Guide to Open Pose 2022". [Online]. Available: https://viso.ai/deep-learning/openpose/ [Accessed November 2022]

[5] Ginés Hidalgo, Zhe Cao, Tomas Simon, Shih-En Wei, Yaadhav Raaj, Hanbyul Joo, and Yaser Sheikh. "Open Pose Documentation" [Online] Available: https://cmu-perceptual-computing-lab.github.io/openpose/web/html/doc/index.html [Accessed November 2022]

[6] Advaith Sridhar, Rohith Gandhi Ganesan, Pratyush Kumar, and Mitesh Khapra. 2020. INCLUDE: A Large Scale Dataset for Indian Sign Language Recognition. In Proceedings of the 28th ACM International Conference on Multimedia (MM '20). Association for Computing Machinery, New York, NY, USA, 1366–1375. https://doi.org/10.1145/3394171.3413528

[7] J. Brownlee, "LSTMs with Python," Machine Learning Mastery, [Online]. Available: https://machinelearningmastery.com/lstms-with-python/

[8] "Hand Landmark Model," Google Developers, [Online]. Available: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker

[9] P. Thakkar, "Python – Facial and Hand Recognition using Mediapipe Holistic," GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/python-facial-and-hand-recognition-using-mediapipe-holistic/

[10] V. Ramakrishnan, "Mediapipe Holistic: Simultaneous Face, Hand, and Pose Prediction," Google AI Blog, [Online]. Available: https://ai.googleblog.com/2020/12/mediapipe-holistic-simultaneous-face.html

[11] R. Singh, "Real-Time Pose Estimation in Webcam using OpenPose Python 2.3 & OpenCV," Medium, [Online]. Available: https://medium.com/pixel-wise/real-time-pose-estimation-in-webcam-using-openpose-python-2-3-opencv-91af0372c31c

[12] V7 Labs, "Video Recognition: Overview and Tutorial," V7 Labs Blog, [Online]. Available: https://www.v7labs.com/blog/video-recognition-overview-and-tutorial