American Sign Language Recognition with Text-To-Speech

A major project report submitted in partial fulfillment of the requirement for the award of degree of

Bachelor of Technology

in

Computer Science & Engineering / Information Technology

Submitted by

Anurag Kumar (201367)

&

Suditi Rathore (201327)

Under the guidance & supervision of

Dr. Nancy Singla

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, Solan - 173234 (India)

CERTIFICATE

This is to certify that the work presented in the "**American Sign Language Recognition with Text-To-Speech**" project report, which was submitted to the Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat in partial fulfilment of the requirements for the award of the degree of B.Tech in Computer Science and Engineering, is an authentic record of work completed by Anurag Kumar (201367) and Suditi Rathore (201327) during the period from August 2023 to May 2024 under the supervision of Dr. Nancy Singla, Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat.

Anurag Kumar
(201367)

Suditi Rathore
(201327)

Dr. Nancy Singla
Assistant Professor (SG)
Computer Science & Engineering and Information Technology
Jaypee University of Information Technology, Waknaghat

CANDIDATE'S DECLARATION

We hereby declare that the work presented in this report entitled **"American Sign Language Recognition with Text-To-Speech"** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering / Information Technology** submitted in the Department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology, Waknaghat is an authentic record of my own work carried out over a period from August 2023 to May 2024 under the supervision of **Dr. Nancy Singla** (Assistant Professor (SG), Department of Computer Science & Engineering and Information Technology).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

(Student Signature with Date)
Student Name: Anurag Kumar
Roll No.: 201367

(Student Signature with Date)
Student Name: Suditi Rathore
Roll No.: 201327

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

(Supervisor Signature with Date)
Supervisor Name: Dr. Nancy Singla
Designation: Assistant Professor (SG)
Department: Computer Science & Engineering and Information Technology
Dated:

ACKNOWLEDGEMENT

Firstly, I express my heartiest thanks and gratefulness to almighty God for his divine blessing making it possible to complete the project work successfully.

I am really grateful and wish my profound indebtedness to my supervisor **Dr. Nancy Singla** Assistant Professor (SG), Department of CSE, Jaypee University of Information Technology, Waknaghat. Her deep knowledge & keen interest in the field of "**Artificial Intelligence**" helped us to carry out this project. Her endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non- instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respeet the constant support and patience of my parents.

Anurag Kumar
201367

Suditi Rathore
201327

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ASL	American Sign Language
CODA	Child of Deaf Adult
SLR	Sign Language Recognition
ROI	Region Of Interest
CNN	Convolutional Neural Network
LSTM	Long Short Term Memory
RNN	Recurrent Neural Network
MP	Media Pipe
ML	Machine Learning
BGR	Blue → Green → Red
RGB	Red → Green → Blue
CV	Computer Vision

ABSTRACT

Humans need a mechanism through which they can communicate with each other. The group of especially abled people are those with speech or hearing disability. Visual communication is also essential while interacting with mute and deaf people. This is because people without any impairment might find it difficult communicating with such individuals.

For effective conversation between two people with and without disabilities, a system of translating hand signs to text messages is required. It is not easy to get interpreters for this kind of a language. Sign language has been one of the ancient and most raw and natural means of communication even to date, although not many people are fluent in it.

Hence, we use neural networks to develop a real time fingerspelling strategy in line with the American Sign Language. To implement this, created a dataset using OpenCV and Mediapipe. Motivation behind creating our own dataset was the insufficient resources and unavailability of an accurate data.

Deep learning approaches can help reduce communication obstacles. The model identifies the gesture accurately and outputs the corresponding text onto the screen. Deep learning technology helps in pattern reorganization and therefore prediction of further data.

CHAPTER 1: INTRODUCTION

1.1 Introduction

A gesture can be seen as motion of the head, body, or hands expressive of thought, sentiment, or passion; any action or posture intended to express a thought or a feeling, or to emphasize or illustrate what is said. Here image processing as well as computer vision(CV) come into use for sign language recognition. Sign recognition enables computers to interpret actions. It can aid the users in interacting spontaneously with the machine systems through artificial interfaces. Sign language is a mode of communication for disabled people, which involves gestures.

For deaf and mute community sign language the only mode of communication. For them sign language is virtually equivalent of a speech. This is a universal language of the deaf and dumb community, though it is manifested locally as ISL and ASL. They use one or two hands to make some specific hand gestures and they speak no other language than the sign language.

CODA (Child of Deaf Adult(s)), it refers to a person who was raised by one or more deaf parents or legal guardians. a sign of continuous and isolated sign languages where instead of using one-word movements, like traditional sign language, continuous ISL consists of movements that produce a complete sentence. This work, involved an independent ASL-gaze recognition language.

Individuals communicate through diverse ways, this includes behavior like physical signals, facial expression, spoken words, and so on but people with a hearing loss can only use hand-gestures to communication. People who suffer from deafness or poor speech use ‘standard’ sign language understood only by a fellow user.

Sign language is ranked sixth among the languages that are used widely. It encompasses gesticulation, which is one way of passing across information by use of hands. Like in other languages, each region has a form of a sign language peculiar for it.

About 62 million people lives with some type of hearing impairment using about 200 sign languages that have their own distinguishing attributes. It is used by hard-of-hearing and normal hearing people as well. However, not all people know the gestures and signals used in the sign language. Sign language understanding and knowing what its movements entail is something that requires plenty of practice. It takes a long period to learn sign language because there is no teaching or any reliable portable tool for detecting it. As with the development of neural networks and deep learning, there appeared a system which can recognize things or things of the same groups.

This project is centered at developing a recognition model for hand motions as well as putting them together to form a complete language word. Figure 1 illustrates a diverse array of American Sign Language (ASL) vocabulary, showcasing expressions for greetings, farewells, affirmations like "hello," "goodbye," and "yes," alongside various other words.

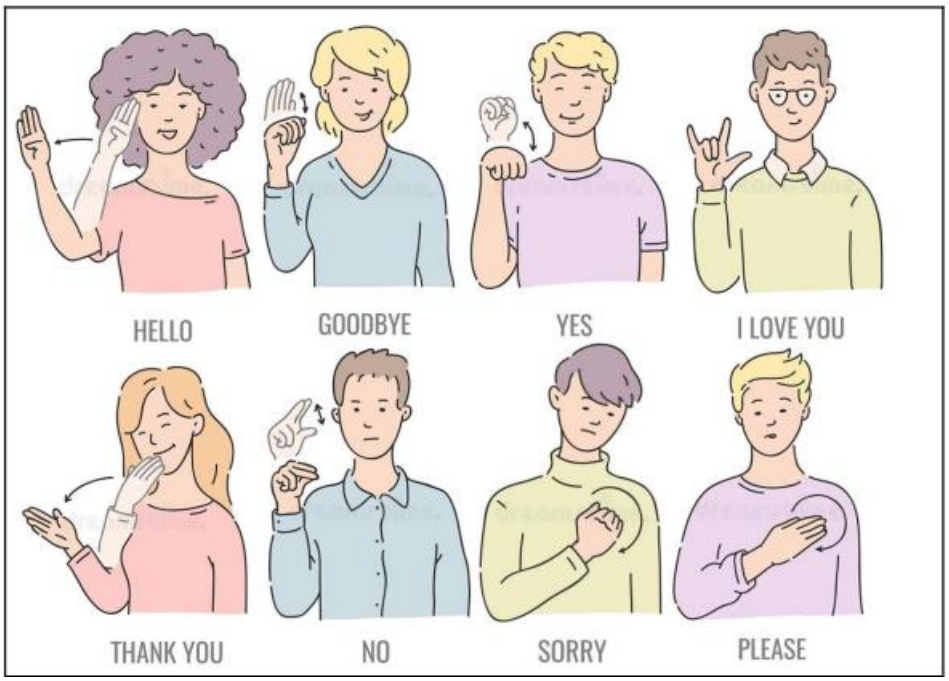

Fig 1. American Sign Languages

1.2 Problem Statement

Communication is a vital part of everyone's day-to-day life. Effective communication serves as the cornerstone of navigating the complexities of daily existence, indispensable for fostering understanding, connection, and successful interactions in all aspects of life. For most people communication seems effortless and easy. But for people with disabilities, it can be a challenge.

Hearing or speech impaired individuals' uses sign language as mode of communication. Most people are not fluent in understanding and recognizing such sign languages. Learning and understanding sign language can be difficult and time consuming. And since there are so many different variations in different sign languages, that it is difficult for a person to understand them.

So, hearing or speech impaired individuals' need a translator, who is proficient in sign language and effectively communicate the ideas to others. A technique is required, which will serve as an intermediate between the people who does and does not understand sign language.

1.3 Objectives

The specific objectives of this project are as follows:

- Creating own dataset using OpenCV, mediapipe holistic by extracting key-points from the data for training and testing of deep learning model.

- Developing a deep learning-based LSTM (Long Short Term Memory) model on trained pre-processed data for real time sign language detection.

- Evaluating the model's performance on various metrics including accuracy, precision, recall, and F1-score.

1.4 Significance and Motivation of the Project Work

The significance and motivation of an American Sign Language (ASL) project can be multifaceted and impactful. Here are some key points to consider:

1. Accessibility and Inclusivity: ASL is the primary means of communication for many individuals who are deaf or hard of hearing. Developing technology that can accurately interpret and translate ASL can greatly enhance accessibility for this community, allowing them to more effectively communicate with others who may not understand sign language.

2. Empowerment and Independence: By enabling real-time recognition of ASL gestures and signs, the project can empower individuals who use sign language to navigate various aspects of daily life more independently. This includes interactions in educational settings, workplaces, social gatherings, and public spaces.

3. Education and Learning: ASL recognition technology can be integrated into educational tools and platforms to facilitate language learning for both deaf and hearing individuals. It can provide interactive feedback and support for practicing ASL vocabulary and grammar, enhancing the learning experience and promoting greater fluency in sign language.

4. Efficiency and Communication: In scenarios where communication barriers exist between individuals who use sign language and those who do not, ASL recognition technology can serve as a bridge, facilitating smoother and more efficient communication exchanges. This can be particularly valuable in healthcare settings, emergency situations, customer service interactions, and more.

5. Research and Innovation: Advancements in ASL recognition technology contribute to ongoing research efforts in the fields of computer vision, machine learning, and human-computer interaction. By pushing the boundaries of what is possible in ASL

interpretation and translation, the project can inspire further innovation and development in related areas.

6. Social Impact and Awareness: The project raises awareness about the importance of accessibility and inclusivity for individuals with disabilities, promoting empathy, understanding, and social change. It highlights the challenges faced by the deaf and hard of hearing community and demonstrates the potential of technology to address these challenges effectively.

In summary, the significance and motivation of the project lie in its potential to enhance accessibility, empowerment, education, efficiency, research, and social impact for individuals who use American Sign Language as their primary mode of communication.

1.5 Organization

The below report is organized as follows:

Chapter 1: Introduction
This chapter sets the stage by clearly defining the problem the project aims to solve. It outlines the specific goals and objectives of the project and provides insight into the methodology employed to achieve these objectives.

Chapter 2: Literature Study
In this chapter, the project builds upon existing knowledge by conducting a thorough review of relevant literature. It identifies areas where previous research may have left gaps or unanswered questions.

Chapter 3: System development and workflow
This chapter dives into the practical aspects of the project, starting with the identification of requirements and analysis of user needs. It then moves on to discuss the design and architecture

of the project, detailing how different components fit together to form a cohesive system. Additionally, it covers data preparation techniques and the actual implementation process.

Chapter 4: Performance analysis

Here, the project evaluates its effectiveness through rigorous testing and analysis. It describes the strategies used for testing, presents the results obtained, and discusses their implications for the project's overall performance and success.

Chapter 5: Conclusion and future scope

Finally, the project wraps up with a comprehensive summary of its findings and conclusions. It also includes future research and development, highlighting areas where further exploration could lead to significant advancements in the field.

CHAPTER 2: LITERATURE SURVEY

2.1 Related Work

Past studies have underscored the importance of leveraging advanced technologies and artificial intelligence algorithms to predict sign language gestures, thereby aiding individuals with hearing impairments. Despite significant research in Sign Language Recognition (SLR), there exist notable limitations and areas for improvement to better serve the hard-of-hearing community. This section provides a concise overview of recent literature focusing on SLR, particularly utilizing sensor and vision-based deep learning techniques.

A review of existing literature reveals various approaches aimed at tackling gesture recognition in videos through diverse methodologies. For instance, in [1], researchers employed Hidden Markov Models (HMM) to identify facial expressions within video sequences. They coupled this with Bayesian Network Classifiers and Gaussian Tree Augmented Naive Bayes Classifiers to enhance recognition accuracy.

Francois et al. [2] contributed to the field by exploring Human Posture Recognition in Video Sequences through both 2D and 3D methodologies. Their work involved utilizing Principal Component Analysis (PCA) to identify silhouettes captured by a static camera, followed by 3D modeling to characterize posture for recognition. However, this approach introduced intermediary gestures, potentially leading to training ambiguities and reduced prediction accuracy.

Similarly, Kumud et al. [4] delved into Continuous Indian Sign Language Recognition, proposing a systematic framework. Their methodology encompassed frame extraction from video data, preprocessing, key frame extraction, feature extraction, recognition, and optimization stages. Preprocessing involved converting videos into sequences of RGB frames with uniform dimensions. Skin color segmentation using HSV facilitated the extraction of skin

regions, subsequently converted to binary form. Key frames were then identified by computing gradients between frames.

Features were extracted from the identified key frames using oriental histograms. Classification was carried out utilizing various distance metrics, including Euclidean Distance, Manhattan Distance, Chessboard Distance, and Mahalanobis Distance.

.

Table 1: Existing Literature Review

S. No.	Paper Title [Cite]	Journal/ Conference (Year)	Tools/ Techniques/ Dataset	Results	Limitations
1.	SignRing: Continuous American Sign Language Recognition Using IMU Rings and Virtual IMU Data	ACM on Interactive Mobile Wearable and Ubiquitous Technologies (2023)	IMU dataset	95.58%.	Video's lower frame rate compared to wearable IMU sensors makes it challenging to achieve sensitivity and stability of IMU sensors.
2.	Sign Pose-based Transformer for Word-level Sign Language Recognition	IEEE/CVF Winter Conference on Applications of Computer Vision Workshops (2022)	LSA64 dataset	63.18%	The model can be trained using a small portion of human labeled data

					to predict precise labels that need correction.
3.	ML Based Sign Language Recognition System	IEEE International Conference on Innovative Trends in Information Technology (2021)	Large dataset	65%	The model showed an accuracy of only 65% due to less data set.
4.	A Modified LSTM Model for Continuous Using Leap Motion	IEEE Sensors Journal (2019)	IIT Roorkee students enrolled for data collection.	72.3%	The model showed an accuracy of only 72.3% due to less training dataset. It uses sensor for detection.
5.	American Sign Language Alphabet Recognition using Deep Learning	ArXiv Journal, (2019)	NVIDIA K80 GPU	71.68%	They have done the recognition only on the alphabets.
6.	American Sign Language	MECS International Journal of Image,	Own dataset	73.05%	Model is restricted to alphabet and

	Recognition - An Optimal Approach	Graphics and Signal Processing (2018)			number recognition only. Dataset consists of the images of hand and fingers.

2.2 Key Gaps in Literature

While the reviewed studies have made significant contributions to American Sign Language Recognition using deep learning, which also shows certain limitations:

Limited Exploration of Continuous Recognition Challenges: The majority of the reviewed studies focus on particular ASL recognition tasks, such as alphabet or numeral recognition. It can be seen that there are considerable areas for undertaking continuous sequence SLR. Hence, it is very crucial to fill gap, as continuous ASL recognition correlates with "natural signing" and is harder, because sign language expression is a dynamic phenomenon.

Scarcity of Standardized Datasets for Robust Evaluation: Standardized datasets are vital in the benchmark and comparison of a number of ASL recognition models. Some studies have observed that most of the models tend to employ small or selected databases, making it impossible for model generalization. But, there is an important gap in the literature. It cannot be argued that there are standard data set for reliable evaluation hence slowing down the growth of general ASL models suitable for various domains.

Insufficient Exploration of Multimodal Approaches: However, there are gaps in the use of multimodal approaches with a combination of various sensors types or different data modalities.

ASL recognition systems can be improved by integrating data from emerging sources, such as video and IMU. Many things could be said about synergistic exploration of these modalities that have not been discussed in the existing literature.

Limited Consideration of Real-world Challenges: There are a number of studies that have pointed out issues associated with low video frame rates, insufficiently labeled data, and lack of diversity in dataset available. However, hardly anything is known about the actual difficulties faced with by ASL system recognition systems in practical conditions that include low light, various users, and noisy surroundings. It is important to address all the mentioned issues above for success on the field.

CHAPTER 3: SYSTEM DEVELOPMENT

3.1 Requirements and Analysis:

Gesture recognition holds immense potential across various domains, from controlling virtual environments to facilitating communication for individuals using sign language. In this machine learning project focused on American Sign Language (ASL) detection, our aim is to develop a real-time tool using the Mediapipe framework and TensorFlow within OpenCV and Python. By harnessing the capabilities of these technologies, we seek to empower users by providing accurate and efficient recognition of ASL gestures. Through this project, we aspire to contribute to the advancement of accessibility and communication technologies, ultimately enriching the lives of individuals who rely on sign language for expression and interaction.

OpenCV

OpenCV is a real-time computer vision and image processing framework built on C/C++. But we will be going to use it in Python with the help of OpenCV python package.

TensorFlow

TensorFlow is an open-source machine learning framework developed by Google, designed to make it easier to build and deploy machine learning models. In an American Sign Language (ASL) project, TensorFlow can be a valuable tool for several reasons.

It provides the necessary tools, flexibility, and support for building robust and efficient ASL recognition systems, making it a compelling choice for developers working on ASL projects.

MediaPipe

The system should be able to process various forms of perceptual data pipelined through the MediaPipe hybrid open source architecture. The use of ML in real-time hand tracking and gesture detection is achieved through the meticulous process. It offers more efficient hand and

finger tracking by providing accurate sign gestures' identification via media Pipe Holistic model we extracted those points from pose of the chest, head, and hands.

MediaPipe holistic pose landmarks for body:

Blaze Pose determines location of persons Areas of Interest (ROI: s) within a particular frame by inferring about thirty-three three-dimensional landmark points in the form of x, y, and z co-ordinates within an input image/video. Progressively, the division masks and pose landmark employ the ROI - cropped frame as input in identifying poses. It is precise as it well identifies essential places and matches an SLR machine.

MediaPipe holistic pose landmark for hands:

MediaPipe Holistic, inside a frame, use two models at a time to estimate about 21 3D hand landmarks consisting of x, y and z coordinates to produce desired output.

i. Palm detection model.
ii. Hand key point localization model.

At first it is applied to a single-shot detector called Blaze Palm. Consider a considerable number of inputs in terms of size (within that of the hand). This detector, to minimize palm detection time complexity, is supported by the MediaPipe. It rather concentrates on superfluous bodies. This method is applied to the whole image and produces a focused bounding rectangular area marking out the rigid parts like the palm or fist for detection.
The second model make use of the palm detection output towards carrying out hand point's localization.

This produced three possible outputs as follows:

- 21 hand knuckle points in a 2D or 3D space.
- Probability of hand flag indicating hand presence within input images.
- Classification of left and right hand as binary.

The below Figure 2 shows a brief overview of how MediaPipe holistics works.

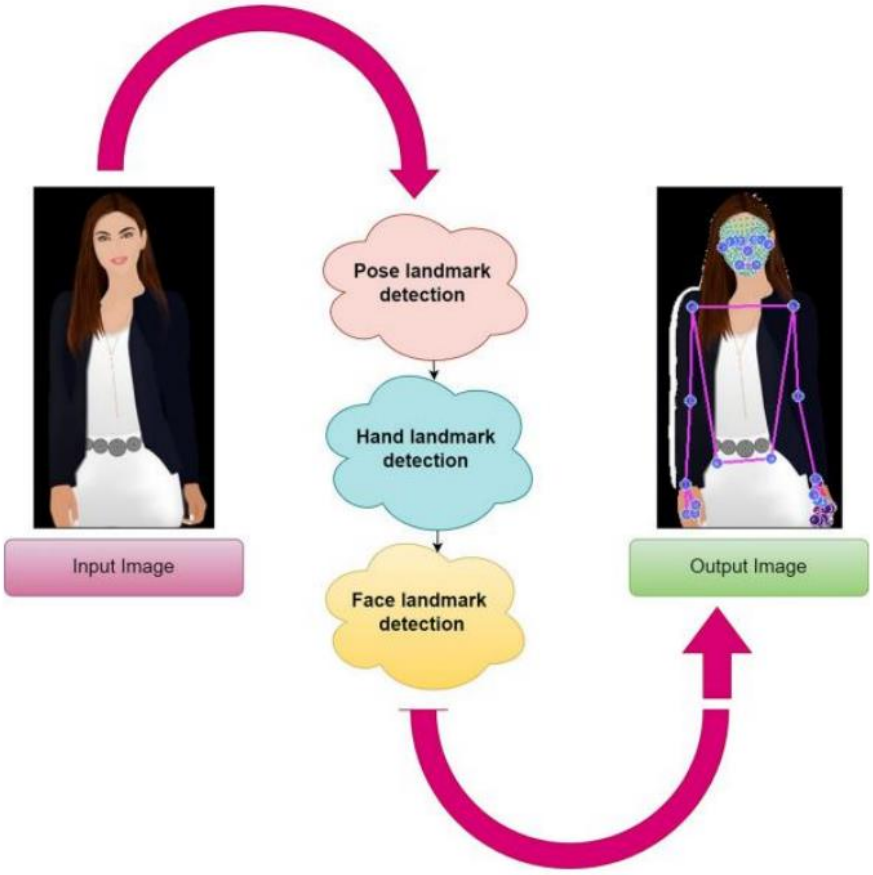

Fig 2. Overview of MediaPipe Holistics

Mediapipe Holistic Model

Custom machine learning framework are designed by google MediaPipe. It is a very simple Frame which has been developed as Open Source, and also works on cross platform. Face detection and tracking is part of pre-trained media pipe ML solutions, among others like hand detection, measurement, object detection and so forth. It is a reliable approach of tracking the hand and fingers. It uses for thirty-three dimensional (21 3D) locality to recognize ML hand marking made into a single frame.

Holistic model for hand detection using mediapipe:

Mediapipe provides a wide range of options. The holistic model extracts 543 individual landmarks in overall (468- face, 33- pose and 21- hand).

With this, we use the mediapipe holistic model that is within the mediapipe python package. Therefore, we made a two-parameter-function (mediapipe detection).

1. Image: The image upon which it does the detection.
2. Model: Holistic model with mediapipe for detection. The function transforms the image from BGR to RGB for the model process function, which then saves the output. It converts it again into BGR, saving the result and returning the image together with it.

Defining another function (draw styled landmarks) that uses parameters from the previous functions. Using this function, we use these landmarks to help us see how to detect hands in real time. Extract Key-points: Following successful real-time hand detection, we get X. Y, and Z coordinate of the key points from their detection results (mediapipe detection function that was discussed earlier).

We construct a separate function called (extract key points) that takes the detection data as an input to obtain the coordinates. It has the holistic model's key point's data, which is a concatenation of all the arrays containing the data. This function works during data-collection and convert a given sentence from AI written to human written

Figure 3-4 shows the extracted key points in a figure. These extractions are for all hand, face, and body.

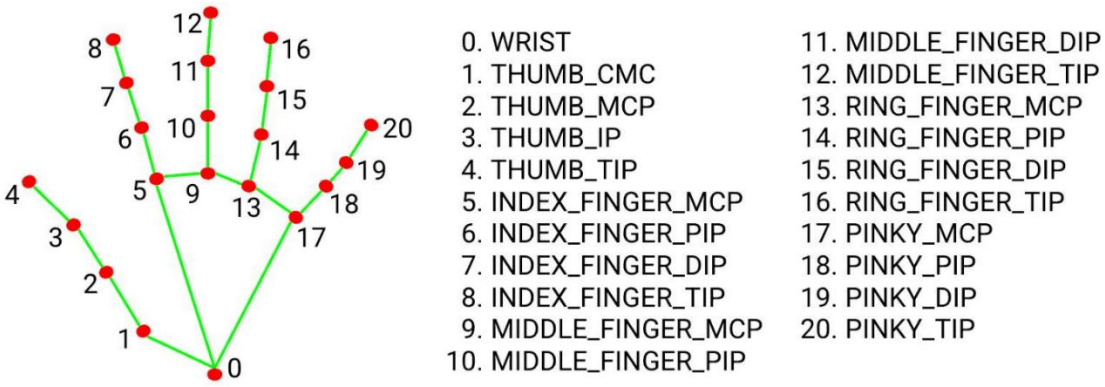

Fig 3. The hand_landmarks model of Mediapipe

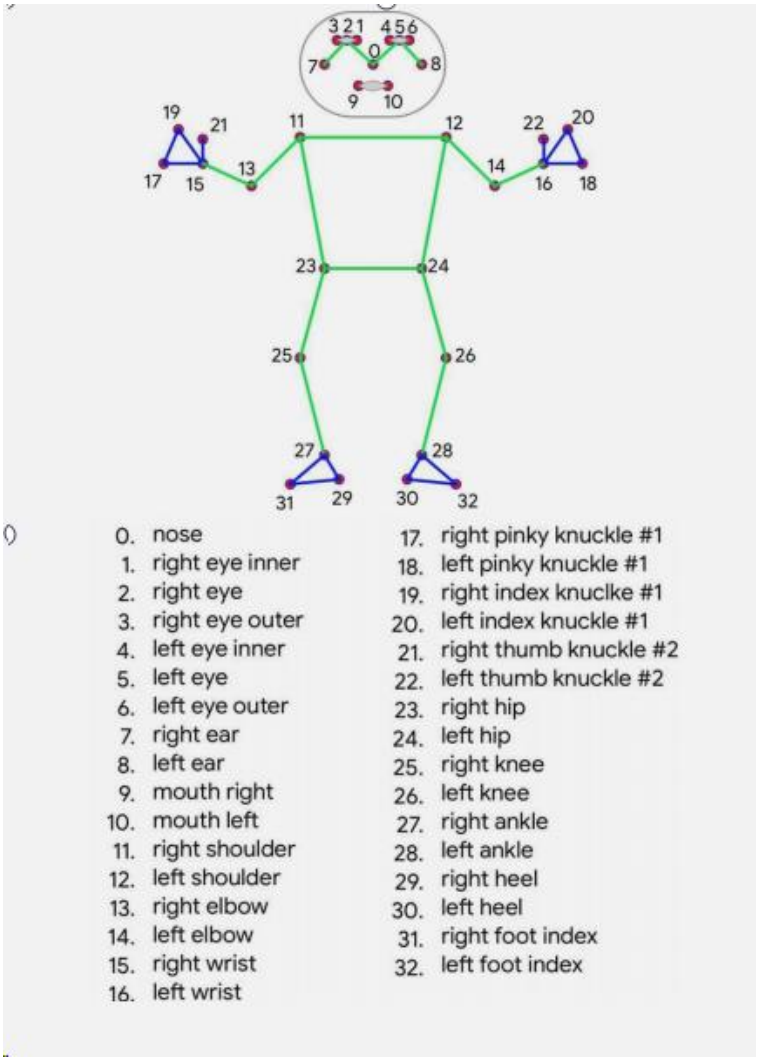

Fig 4. The pose_landmark model of Mediapipe

Implementation of LSTM Long short-term memory

LSTM (Long Short Term Memory), a variant of RNN, specializes in handling sequential or time-series data. Unlike traditional RNN units, LSTM units are more intricate, featuring gates that regulate data flow within each unit. The recurrent loop is pivotal in LSTM architecture, enabling the retention of past information in its internal state memory. This mechanism allows for the extraction of both spatial and temporal information from the data, enhancing its capability to analyze sequences effectively.

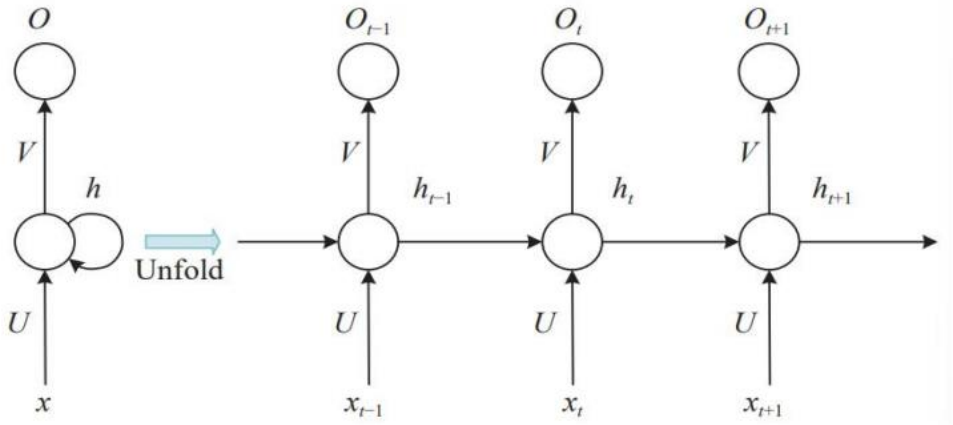

Fig 5. The architecture of recurrent neural network

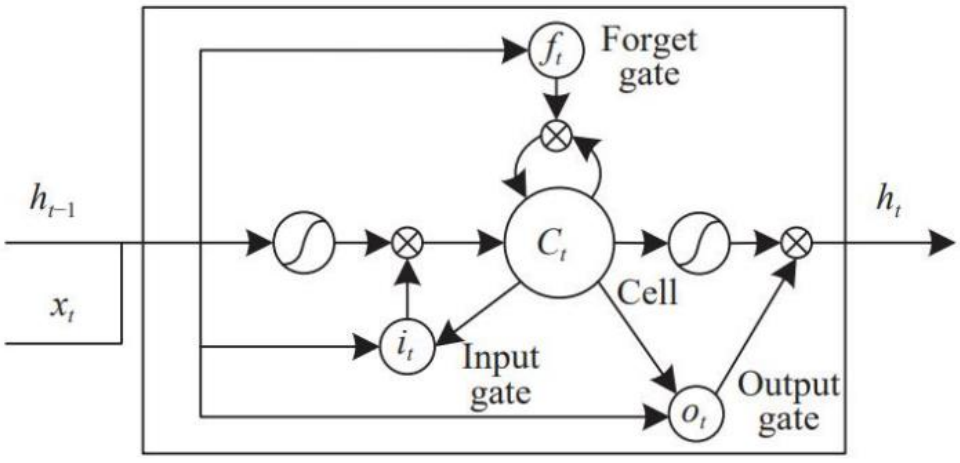

Fig 6. The architecture of the LSTM block

A single LSTM model has architecture containing 3 gates and 1 memory cell (figures 5 and 6).

i. The forget gate expunges information which should not be retained in the state of cell. It states that when knowledge ceases being relevant in terms of a given environment, it might get erased from the mind.
ii. Input Gate, whose purpose is to load the cell stage with vital data. This involves adding new facts and the updating of our working storage state.
iii. Output gate is the one responsible for retrieval of relevant content from present condition of a cell and providing same as output information.

To every gate an appropriate operation's value of a number between 0 and 1 is assigned. Therefore a value equal to zero or one is not information since it results into no transmission of any information between these values. A single consists of four interconnected layers. Figure 7 depicts the repeating module in an LSTM contains four interacting layers.

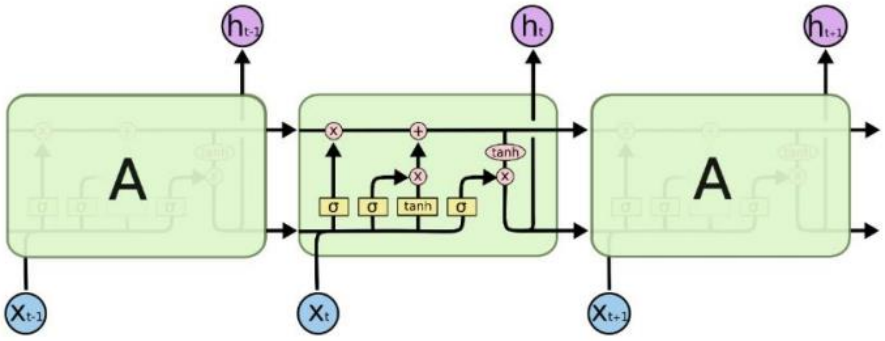

Fig 7. The repeating module in an LSTM contains four interacting layers

We have compiled our various data into individual frames; each frame indicates all the land points that were acquired for different sites. Each frame is stored as a NumPy array. The linear and sequential view is the basis for building an LSTM network. The first four layers have three thick layers. The first six layers receive "sigmoid" type of activation function. In this case, softmax is used for the final dense layer. An optimizer known as ADAM is utilized for stochastic gradient descent.

3.2 Project Design and Architecture

The proposed system addresses the challenges encountered by previous models without compromising efficiency or performance. One significant issue was the non-uniform background and segmentation of hands from the background. This problem was resolved by creating a dataset using Google's Mediapipe solution and the OpenCV library to detect landmarks from the hand. These landmarks were also utilized during real-time detection to ensure accurate results regardless of the environment.

The system consists of eight primary modules: Creating Dataset, Importing dependencies, Extracting key-points, Collecting Key-point sequences, Preprocessing Data and Create labels, Building LSTM Model, Prediction and Evaluation of the Model. The architecture of the proposed ASL Recognition System is illustrated in Figure 8.

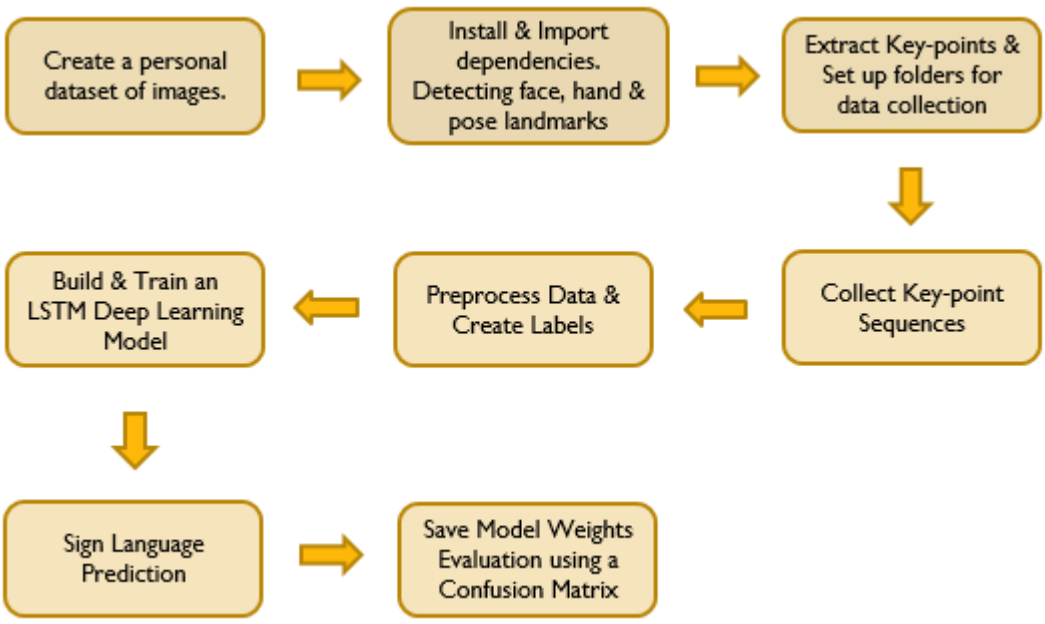

Fig 8. Architecture of the proposed ASL Recognition System

Here's an explanation of each step in the architecture of the proposed American Sign Language (ASL) recognition system:

1. Create a personal dataset of images:

    - This step involves gathering a collection of images or videos depicting ASL gestures. These can include various hand signs representing letters, words, or phrases in ASL. Building a diverse dataset is crucial for training a robust recognition system.

2. Install & Import dependencies. Detecting face, hand & pose landmarks:

    - Here, necessary libraries and dependencies are installed or imported. This includes frameworks like TensorFlow and Mediapipe for deep learning, as well as specific libraries like cv2, numpy, matplotlib and pyplot for computer vision tasks such as detecting faces, hands, and pose landmarks. These landmarks provide crucial information for recognizing ASL gestures accurately.

3. Extract Key-points & Set up folders for data collection:

    - In this step, key points or landmarks from the images or videos in your dataset are extracted. These key points represent specific locations on the face, hands, or body, which are important for recognizing gestures. Additionally, folders are created to organize data collection process, ensuring that images or videos are stored systematically for training and validation.

4. Build & Train an LSTM Deep Learning Model:

    - This is where Long Short-Term Memory (LSTM) deep learning model is constructed and trained. LSTM's are a type of recurrent neural network (RNN) well-suited for sequential data like time series or sequences of key points, making them ideal for capturing temporal dependencies in ASL gestures.

5. Preprocess Data & Create Labels:

    - Before feeding the data into the LSTM model, it needs to be preprocessed. This may involve resizing images, normalizing pixel values, or converting data into a suitable format for training. Additionally, you create labels or annotations for each sample in the dataset, indicating the corresponding ASL gesture.

6. Collect Key-point Sequences:

    - With the dataset prepared and labeled, sequences of key points representing ASL gestures are extracted from the images or videos. These sequences serve as input to the LSTM model during training and prediction.

7. Sign Language Prediction:

    - Once the LSTM model is trained, it can predict the ASL gesture represented by a given sequence of key points. During prediction, the model takes in a sequence of landmarks captured in real-time from a camera feed or static images and outputs the predicted ASL gesture.

8. Save Model Weights Evaluation using a Confusion Matrix:

    - After training, the trained model weights can saved for future use. Additionally, performance of the model using metrics like accuracy and a confusion matrix is evaluated.

By following these steps, a robust ASL recognition system can be developed which is capable of accurately interpreting sign language gestures.

3.3 Data Preparation

The first step is to create folders for data collection. We gathered our own data set for training and testing our deep learning model in the following manner: Using the path.join('data folder name') method, we created a folder for the dataset. We then defined a set of gestures (["bye", "sorry"," I love you", "stop"]) that will be used to train our recognition model. After that we specify the number of images to be collected (30) and the number of frames (30) for each image to be captured. For each of the gesture folder there are subfolders folders as shown in the Figure 9 and Figure 10 below:

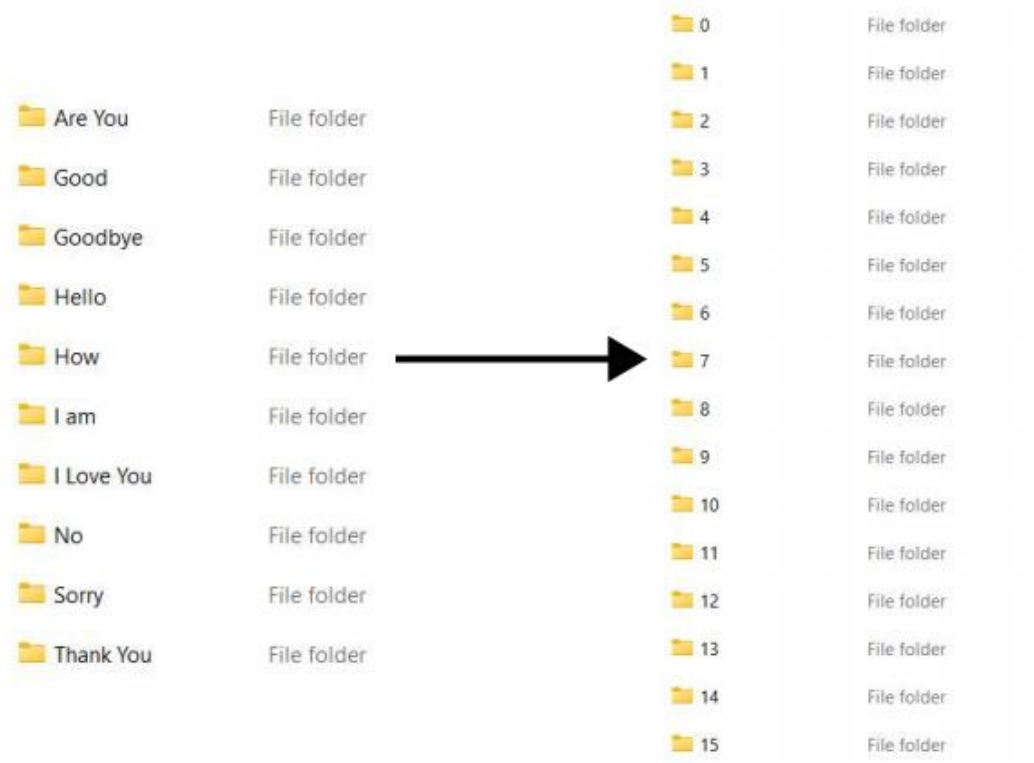

Fig 9. Folder for gesture and corresponding subfolders for each gesture

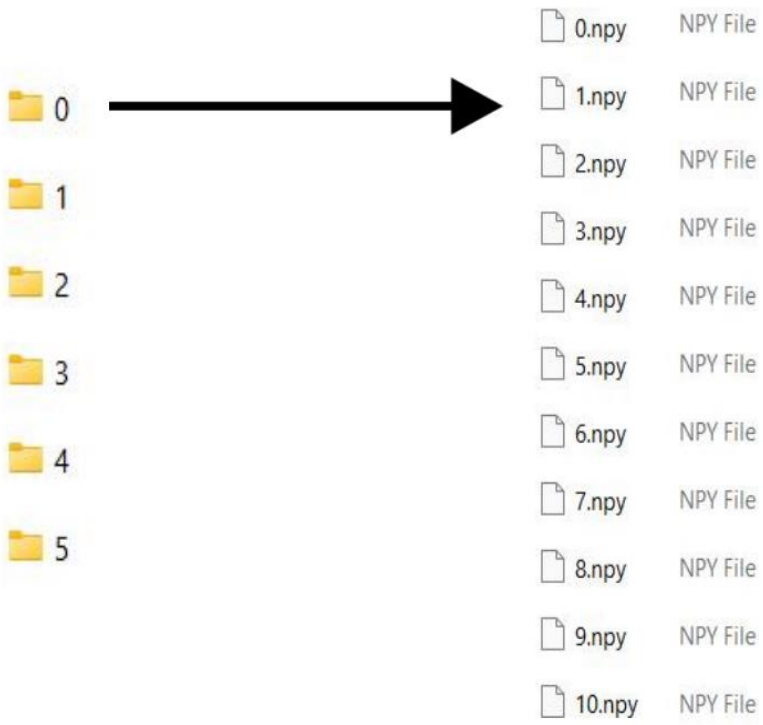

Fig 10. Collection of 30 frames inside each subfolder

Once the data collection phase wraps up, the extracted key points undergo meticulous structuring through data preprocessing. This process involves organizing all gesture key point arrays into a single cohesive NumPy array (X), paired with corresponding labels mapped onto another NumPy array (Y). Subsequently, the categorical function transforms Y into a binary class matrix, facilitating classification. For instance, [1,0,0...] denotes "hello," [0,1,0...] represents "thanks," and [0,0,1,0,0...] signifies "I love you," and so forth. Post-preprocessing, the dataset is partitioned into training and testing subsets using the train-test split function.

To decode sign language, we rely on extracted key-points derived from 30 frames of data, totaling to 301662 key-points to discern specific actions. Unlike other computer vision tasks, action detection involves analyzing a sequence of data rather than individual frames. For our project, we've gathered data for 10 distinct actions, such as "hello," "thanks," and "I love you." Each action comprises 30 videos, with each video containing 30 frames of data. Each frame

encapsulates 1662 landmark values, resulting in a structured dataset of 1030 sequences, each with 30 frames and 1662 landmarks.

We first establish our data path, where our dataset is stored. Subsequently, we define the actions variable, representing the various actions we aim to detect. Through a loop iterating over the number of actions and sequences, we create corresponding folders to organize the data effectively. This meticulous process lays the foundation for training our model to accurately recognize and interpret sign language gestures.

3.4 Implementation:

3.4.I. Install and import the dependencies

```
import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
import time
import mediapipe as mp
```

Fig 11. Importing libraries

In Figure 11 some important libraries are imported for different purposes in the process of developing the project. The incorporation of OpenCV (cv2) facilitates various tasks related to computer vision, providing an array of capabilities in handling images and videos. It uses NumPy to perform efficient numerical operations and array manipulation to improve arithmetic calculations in the project. In turn, the os library can access the operating system facilitating file and directory manipulation. Data visualization requires matplotlib that has got a huge number of plotting features needed to present results graphically.

By integrating time, timing can be controlled with accuracy during evaluations and delays can be injected when needed. Finally, the inclusion of the Mediapipe library shows that the concern is with using ready-made solutions for the hand and pose detection problems to reduce the burden of implementing complicated vision methods. Combined together, they are a set of tools to help with image processing, computation, operating system interaction, and visualization, timing, and specialized hand and poses.

3.4.II. Detect Face, Hand and Pose Landmarks (using MP Holistic)

To begin our project, we first ensure proper access to the webcam using OpenCV. By setting up a video capture, we establish a connection with the webcam device (designated as value 0). We then iterate through each frame captured by the webcam, effectively creating a real-time video feed.

Key functions utilized include:

- cv2.VideoCapture(0): This function initiates access to the webcam, with the device value set to 0.
- cap.isOpen(): This method verifies if the webcam is successfully accessed, facilitating the initiation of a loop.
- cap.read(): To retrieve the current frame from the webcam feed. It returns two values: a return value indicating success and the current frame itself.
- cv2.imshow(): Utilized to display the output frame on the screen.
- cv2.waitKey(): This function waits for a key press from the keyboard, enabling interactive control.
- cap.release(): Releases the webcam device once access is no longer required.
- cv2.destroyAllWindows(): Closes down the frame window, terminating the program's execution smoothly.

These operations collectively establish a foundational setup for further processing, such as face, hand, and pose landmark detection using MP Holistic, facilitating the development of our real-time gesture recognition system.

```
vid = cv2.VideoCapture(0)

while(True):

    # Capture the video frame
    # by frame
    ret, frame = vid.read()

    # Display the resulting frame
    cv2.imshow('frame', frame)

    # the 'q' button is set as the
    # quitting button you may use any
    # desired button of your choice
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# After the loop release the cap object
vid.release()
# Destroy all the windows
cv2.destroyAllWindows()
```

Fig 12. Video Capture

Figure 12 depicts the code that creates a real-time video capture window from the webcam, allowing the user to view the live feed until they decide to quit by pressing the 'q' key.

Next, we will set up Mediapipe Holistic for our detection tasks. To begin, we need to capture the image and convert it from BGR to RGB format. This conversion is crucial as Mediapipe processes images in RGB format. To optimize memory usage, we set the image to un-writeable during detection. After performing the detection, we reset the image to writable and convert it back to BGR format for further processing or display. This process is encapsulated in the following function:

def mediapipe_detection(image, model)

This function handles the necessary image conversions and performs the detection using the Mediapipe model, ensuring efficient and accurate processing.

By default, OpenCV captures video feed in the BGR color format. However, for Mediapipe to perform detections, the image must be in RGB format. We achieve this conversion using OpenCV's “cv2.cvtColor()” function, which changes an image from one color space to another. Specifically, we convert the BGR image to RGB format, ensuring the correct order of channels.

Once the image is converted to RGB, we can proceed with Mediapipe detection. After detecting the keypoints, the next step is to visualize these landmarks on the image. Currently, the landmarks are not rendered on the frame. To visualize them, we define the following function:def draw_landmarks(image, results)

mp.drawing.draw_landmarks()

This function will draw the detected keypoints on the image, allowing us to see the landmarks directly on the video feed.

```
# Holistic detetction API (Hands, Face and Pose)
mp_holistic = mp.solutions.holistic
# Drawing Utilities for keypoint and landmark detection
mp_drawings = mp.solutions.drawing_utils
```

Fig 13. Using Holistic API’s

This python code snippet in Figure 13 uses the holistic API from the media pipe library to detect hands, head, and body pose in pictures or video frames. mp_holistic identifies several body parts sequentially in real-time while mp_drawing draws points such as key points and landmarks on input data. An integrated strategy offers a good way of examining the different manners of looking at one’s pose and expression, which include gestures and fitness.

```python
def mediapipe_detection(image, model):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image.flags.writeable = False
    results = model.process(image)
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    return image, results
```

Fig 14. Mediapipe_detection function

The Python function ‘mediapipe_detection’ used in Figure 14 takes an image and a pre-initialized Mediapipe model as input. First, it translates the input image from BGR to RGB color space – the expected input of Mediapipe models. To better achieve efficiency at inference, it temporarily sets the writeable attribute of the image to false. Afterwards, the function processes the image with the supplied model and gets the detection results.

```python
def draw_landmarks(image, results):
    # Face Landmarks
    mp_drawings.draw_landmarks(image,
                               results.face_landmarks,
                               mp_holistic.FACEMESH_TESSELATION,
                               mp_drawings.DrawingSpec(color=(6, 70, 99), thickness=1, circle_radius=1),
                               mp_drawings.DrawingSpec(color=(236, 255, 101), thickness=1, circle_radius=1))
    # Pose Landmarks
    mp_drawings.draw_landmarks(image,
                               results.pose_landmarks,
                               mp_holistic.POSE_CONNECTIONS,
                               mp_drawings.DrawingSpec(color=(6, 70, 99), thickness=1, circle_radius=1),
                               mp_drawings.DrawingSpec(color=(236, 255, 101), thickness=1, circle_radius=1))
```

Fig 15. Draw_landmarks function

The draw_landmarks Python function in figure 15 uses the MediaPipe library to detect facial and pose landmarks on an image with supplied outputs. It indicates the points and links via ‘mp_holistic’. “FACEMESH_TESSELATION” for face landmarks and ‘mp_holistic’. “POSE_CONNECTIONS” for pose ones. These landmarks are subsequently superimposed on the input image via ‘mp_drawings.draw_landmarks’ in this function. Colour, thickness, and circle radius will be defined by two sets of Drawing Spec objects that are used for drawing the

landmarks in order to increase visual perception of the both faces and body positions. It has become useful over time for graphical depiction of critical areas in application like faces detection and posture identification.

```
# Left Hand Landmarks
mp_drawings.draw_landmarks(image,
                           results.left_hand_landmarks,
                           mp_holistic.HAND_CONNECTIONS,
                           mp_drawings.DrawingSpec(color=(6, 70, 99), thickness=1, circle_radius=1),
                           mp_drawings.DrawingSpec(color=(236, 255, 101), thickness=1, circle_radius=1))
# Right Hand
mp_drawings.draw_landmarks(image,
                           results.right_hand_landmarks,
                           mp_holistic.HAND_CONNECTIONS,
                           mp_drawings.DrawingSpec(color=(6, 70, 99), thickness=1, circle_radius=1),
                           mp_drawings.DrawingSpec(color=(236, 255, 101), thickness=1, circle_radius=1))
```

Fig 16. Implementing function for each hand

In Figure 16 the code further extends the visualization capabilities, drawing out the landmarks for the left hand and right hand on the input image. It specifies landmark connections for the left hand using ‘mp_holistic’. “HAND_CONNECTIONS”, and equally does so for the right hand. It provides the ‘mp_drawings-draw_landmarks’ function that superimposes the extracted hand’s landmarks over the image thus improving the depiction of moving hands and gestures in a video. The settings of ‘DrawingSpec’ are related to color, thickness and radius of the drawn landmarks, making it easy to interpret the visualization. Therefore this functionality has been of valuable use in the fields of sign language interpretation, hand gesture recognition and smart interactive interfaces.

```
cap = cv2.VideoCapture(0)
with mp_holistic.Holistic(min_detection_confidence=0.7, min_tracking_confidence=0.7) as holistic:
    while cap.isOpened():
        ret, frame = cap.read()
        image, results = mediapipe_detection(frame, holistic)
        draw_landmarks(image, results)

        cv2.imshow('OpenCV Feed', image)
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
cap.release()
cv2.destroyAllWindows()
```

Fig 17. Capturing

The Python code in Figure 17 takes a live feed from the default camera with the help of opencv. It combines the MediaPipe holistic model of holistically detecting facial features, skeleton position, and finger positions.

It is always reading frame after frame from the camera and processing them through the mediapipe_detection function, drawing landmarks with draw_landmarks. “OpenCV Feed” is a window where one can display the processed frames obtained as output.

When you press ‘q’, the script will exit and then, release the video capture using cap.release(), and close OpenCV window/s by performing cv2.destroyAllWindows(). The integration shows actual time visualization of whole landmarks onto the video stream as showed in Figure 18.

```
In [7]: draw_landmarks(frame, results)

In [8]: plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
Out[8]: <matplotlib.figure.AxesImage at 0x162a948b490>
```

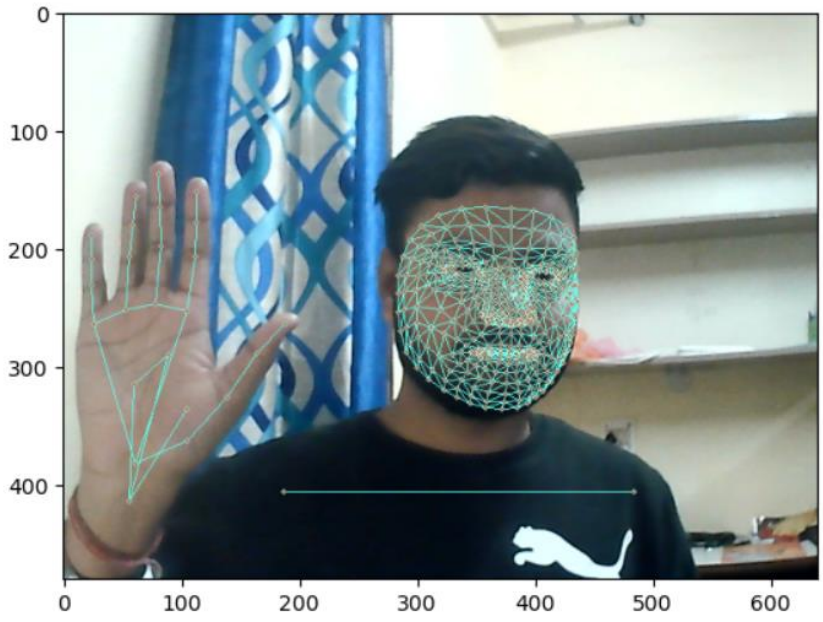

Fig 18. Detection of hand and face

3.4.III. Extract Keypoint Values

Up to this point, we have successfully obtained landmarks for the left hand, right hand, face, and body pose. Our next task is to extract these landmarks in a robust manner, ensuring consistency even when some values are missing. To achieve this, we will concatenate the landmarks into a NumPy array. If any values are missing at a given time, we will substitute them with a NumPy array of zeros, maintaining the same shape but filled with zero values.

The input data for our action detection model consists of a series of 30 arrays, each containing 1662 values, resulting in a shape of (30, 1662). Each array represents the landmark values from a single frame. We will create placeholder arrays for the left hand, right hand, face, and pose. By extracting the landmarks for each keypoint, we will update a single flattened array. This process is encapsulated in the following function:

def extract_keypoints(results)

This function ensures that all extracted keypoints are consolidated into one array, accommodating any missing data by using zero-filled arrays to preserve the overall structure.

```
len(results.right_hand_landmarks.landmark)
```

```
21
```

```
pose = []
for res in results.pose_landmarks.landmark:
    test = np.array([res.x, res.y, res.z, res.visibility])
    pose.append(test)
```

```
pose = np.array([[res.x, res.y, res.z, res.visibility] for res in results.pose_landmarks.landmark]).flatten() if results.po
face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten() if results.face_landmarks els
lh = np.array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.landmark]).flatten() if results.left_hand_landm
rh = np.array([[res.x, res.y, res.z] for res in results.right_hand_landmarks.landmark]).flatten() if results.right_hand_lan
```

```
face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten() if results.face_landmarks els
```

```
def extract_keypoints(results):
    pose = np.array([[res.x, res.y, res.z, res.visibility] for res in results.pose_landmarks.landmark]).flatten() if result
    face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten() if results.face_landmarks
    lh = np.array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.landmark]).flatten() if results.left_hand_l
    rh = np.array([[res.x, res.y, res.z] for res in results.right_hand_landmarks.landmark]).flatten() if results.right_hand_
    return np.concatenate([pose, face, lh, rh])
```

```
result_test = extract_keypoints(results)
```

```
result_test
```

```
array([ 0.49308124,  0.48694369, -0.82234895, ...,  0.07018319,
        0.48987353, -0.04031748])
```

```
np.save('0', result_test)
```

```
np.load('0.npy')
```

```
array([ 0.49308124,  0.48694369, -0.82234895, ...,  0.07018319,
        0.48987353, -0.04031748])
```

Fig 19. Extracting Key-points

This code snippet in Figure 19 contains information that you can use to retrieve key point values for the output of a holistic detection model supplied by MediaPipe. It comprises script of an

extract key-points function (results). The next step entails merging the resulting key-points to form a single NumPy array.

- For body pose (`pose`), each landmark is represented by four values: x, y, z, and visibility.
- For the face (`face`), each landmark is represented by three values: x, y, and z.
- Each hand (`lh` and `rh`) has a total of 21 key-points flattened to arrays containing x, y, z coordinate values.

This is a concatenated array of all these key-points. This script also exemplifies how these key-points get pulled out from the 'results' array, then saved as a NumPy file called "0.npy". As an alternative, the resulted arrays can be saved on the computer memory by the use of `np.save` to read and `np.load`, which load it to read. This is advantageous as it makes it easy to conduct analysis or train the model with the acquired information. When you save it, load the file later by using of `np.load('0.npy')` just like you are doing in your code. These extracted keypoints can come in handy for performing gesture recognition, pose estimation, and even in applications that entail spatial arrangement of both face and body characteristics.

3.4.IV. Collect Keypoint Sequences

We will incorporate breaks between each video collection session. These breaks allow us to reset and reposition ourselves, ensuring that each action sequence is captured from start to finish accurately. After collecting the data, the frames are stored as NumPy arrays at the specified data path location. The following examples are illustrated in Figures 20 to 23.

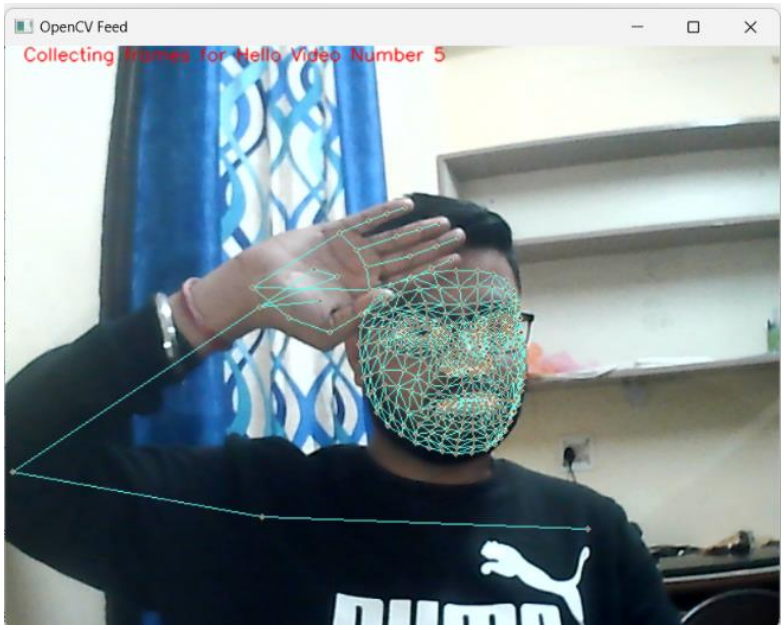

Fig 20. Collecting frames for “Hello” word

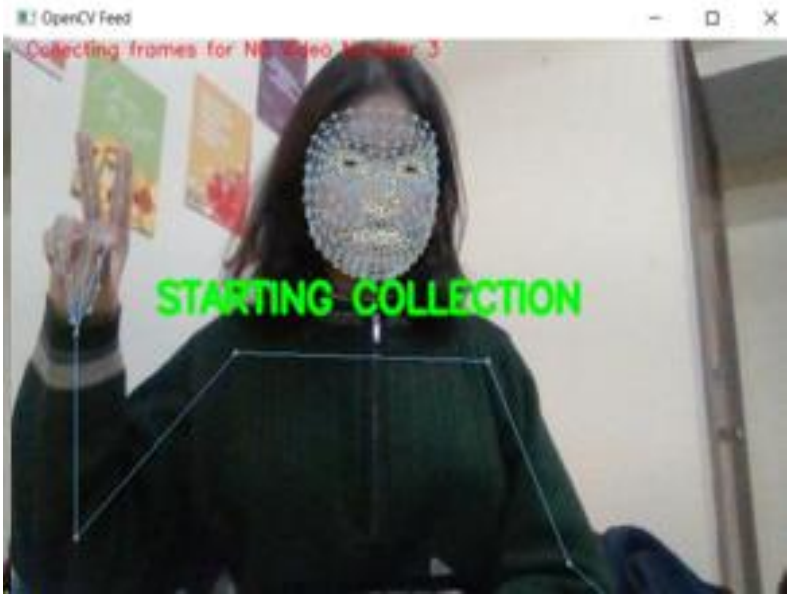

Fig 21. Fig Collecting frames for “ No”

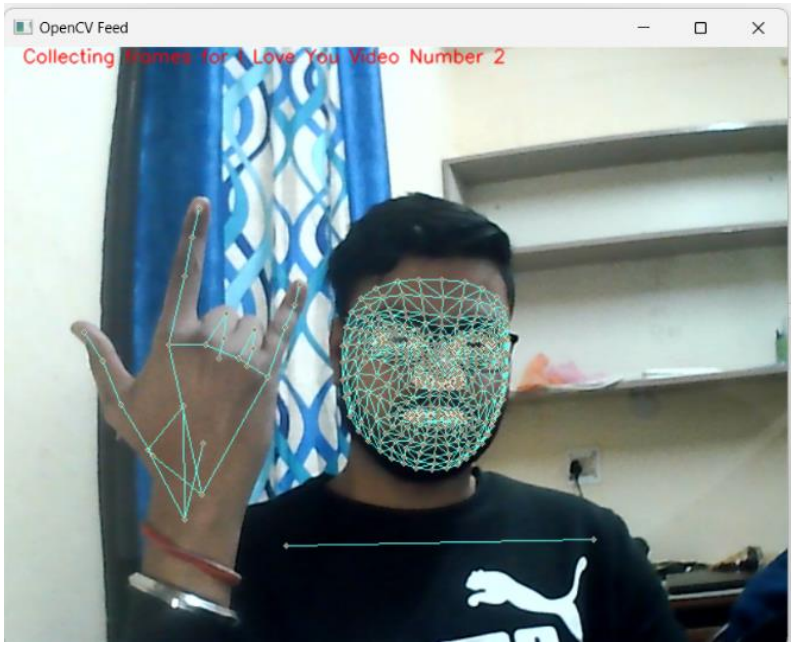

Fig 22. Collecting frames for “I love you”

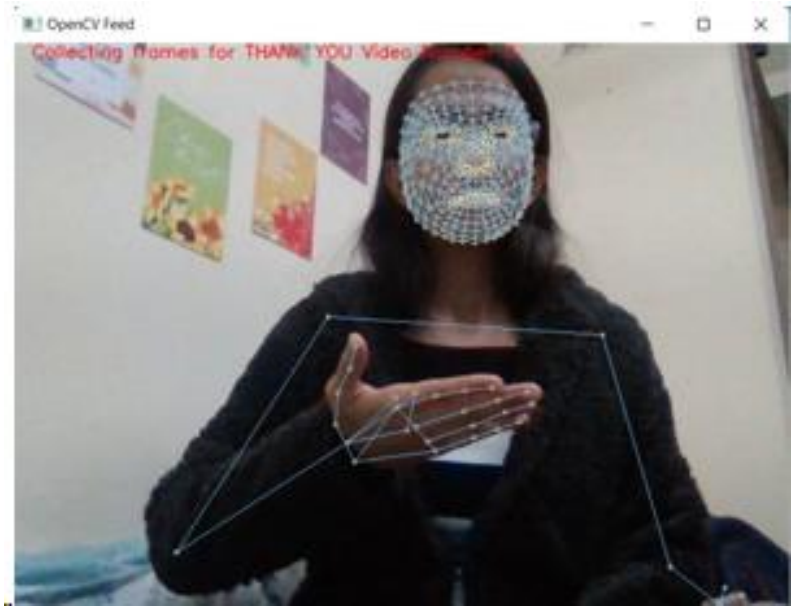

Fig 23. Collecting frames for “Thank You”

3.4.V. Pre-process Data and Create Labels

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
```

```
label_map = {label:num for num, label in enumerate(actions)}
```

```
label_map
```

```
{'Hello': 0,
 'Thank You': 1,
 'I Love You': 2,
 'Are You': 3,
 'Good': 4,
 'Sorry': 5,
 'Goodbye': 6,
 'How': 7,
 'No': 8,
 'I am': 9}
```

Fig 24. Pre Process Data

In figure 24, code snippet facilitates the preprocessing of ASL dataset by assigning numerical labels to each gesture class, which is a necessary step before training a machine learning model likely for American Sign Language (ASL) recognition based on the provided labels.

1. Import Libraries: The code imports necessary libraries:

    - train_test_split from sklearn.model_selection : This function splits the dataset into training and testing sets.
    - to_categorical from tensorflow.keras.utils : This function converts integer labels into one-hot encoded vectors.

2. Label Mapping:

    - The label_map dictionary is created to map each ASL gesture label to a numerical value. Each label is associated with a unique number, starting from 0.

This mapping is essential for converting text labels into numerical values that can be used in the model.

Next, we will create a label map to represent each of our different actions, and begin to consolidate and structure our data. This involves creating a comprehensive array that contains all of our collected data.

Ultimately, we will have 10 actions, each with 30 sequences, and each sequence will consist of 30 frames containing 1662 keypoint values.

First, we will create two blank arrays:

1. sequences - representing our feature data (X data).
2. labels - representing our label data (Y data).

We will use these features to train a model that can learn the relationship between the features and their corresponding labels. We will loop through each action and each sequence within those actions.

We will then create a blank array called window to store all the frames for a particular sequence. By using numpy.load(), we will load each frame into this window. The sequences array will ultimately contain 10 actions, each with 30 sequences, and each sequence will comprise 30 frames. We will store these sequences in a NumPy array for easier manipulation. Following this, we will preprocess the data and partition it for training and testing. We will use the train_test_split() function with test_size=0.05, meaning 5% of our data will be reserved for testing.

Sequence and label array is created as shown in Figure 25.

```
sequences, labels = [], []
for action in actions:
    for sequence in np.array(os.listdir(os.path.join(DATA_PATH, action))).astype(int):
        window = []
        for frame_num in range(sequence_length):
            res = np.load(os.path.join(DATA_PATH, action, str(sequence), "{}.npy".format(frame_num)))
            window.append(res)
        sequences.append(window)
        labels.append(label_map[action])
```

```
np.array(sequences).shape
```

```
(300, 30, 1662)
```

```
np.array(labels).shape
```

```
(300,)
```

```
X = np.array(sequences)
```

```
X.shape
```

```
(300, 30, 1662)
```

```
y = to_categorical(labels).astype(int)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)
```

```
y_test.shape
```

```
(15, 10)
```

Fig 25. Create Labels

3.4.VI. Build and Train LSTM Deep Learning Model

To train our LSTM neural network, we will use TensorFlow and its Keras API. First, we need to import several dependencies:

1. Sequential module: Allows us to build a sequential neural network.
2. LSTM layer: Provides a temporal component for our neural network, enabling us to perform action detection.
3. Dense layer: A standard fully connected layer.
4. TensorBoard: Enables logging and monitoring of our model's training process.

Next, we will create a log directory and set up TensorBoard callbacks for tracking the training process. We will then build the neural network architecture, compile the model, and fit it to our data.

We will use the Adam optimizer and the categorical crossentropy loss function, appropriate for our multi-class classification model. After compiling the model, we will proceed to fit and train it. By following these steps, we ensure our LSTM neural network is properly configured and optimized for action detection.

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

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

```
model.fit(X_train, y_train, epochs=750, callbacks=[tb_callback])
```

Fig 26. Building LSTM Model

In figure 26, the code is used to build a basic LSTM model which takes an input and provides an output accourdingly. LSTM stands for long short term memory which helps to retain the longer session of data and eliminate the problem faced by RNN. Vanishing gradient problem and long term dependence issues are solved using LSTM.

3.4.VII. Make Sign Language Predictions

Making predictions using LSTM model, extracting the predicted actions, and comparing them with the ground truth actions for the fifth sample in the test data. The code snippet for making sign language prediction is given in Figure 27.

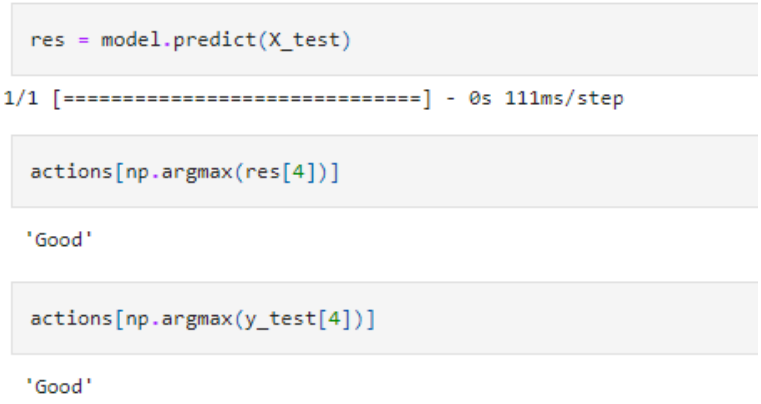

Fig 27. Making Predictions

res = model.predict(X_test) : Using LSTM model to make predictions on the test data (X_test). The result is stored in the variable res. So, res contains the predicted outputs for each sample in the test data.

actions[np.argmax(res[4])] : Selecting the action corresponding to the highest probability prediction for the fifth sample in the test data (res[4]). np.argmax() returns the index of the maximum value in the array res[4]. actions likely contains the labels or actions that the model can predict, and the selected action is retrieved using this index.

actions[np.argmax(y_test[4])] : Selecting the action corresponding to the highest probability ground truth label for the fifth sample in the test data (y_test[4]). Similar to the previous code, it retrieves the index of the maximum value in the ground truth array y_test[4] and selects the corresponding action from the actions array.

3.4.VIII. Save Model Weights

For saving and loading the weights of the LSTM model using the Keras library in Python:

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

```
# model.load_weights('action.h5')
```

Fig 28. Save Model Weights

model.save('action.h5'): It save the weights of the model to a file named 'action.h5'. This file contain the model's architecture as well as the learned weights.

model.load_weights('action.h5') : It load the weights saved in the file 'action.h5' back into the model. This assumes that the model architecture is already defined and matches the one used when saving the weights.

Figure 28 shows the code which are useful for saving the trained state of a machine learning model after training, allowing you to later load the model with the same configuration and weights for inference or further training without needing to retrain the model from scratch.

3.4.IX. Evaluation using Confusion Matrix and Accuracy

After training our LSTM neural network, we will evaluate its performance using a confusion matrix and accuracy metrics.

Confusion Matrix:

A confusion matrix is a table that allows visualization of the performance of a classification model. It summarizes the actual class labels against the predicted class labels.

Accuracy:

Accuracy measures the proportion of correctly classified instances among all instances. It is calculated as the ratio of the sum of true positive and true negative predictions to the total number of predictions. Mathematically, accuracy can be expressed as:

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

Figure 29 shows the python code snippet of Evaluation using a Confusion Matrix and Accuracy.

- model = tf.keras.models.load_model('action.h5') : This line loads a pre-trained model from the file 'action.h5'. This assumes that the model architecture and weights are saved in this file.

- yhat = model.predict(X_test) : This line uses the loaded model to make predictions on the test data (X_test). The predictions are stored in the variable yhat.

- ytrue = np.argmax(y_test, axis=1).tolist() : It converts the true labels (y_test) from one-hot encoded format to a list of integers using np.argmax(). This is necessary for calculating the confusion matrix and accuracy score.

- yhat = np.argmax(yhat, axis=1).tolist() : Similarly, it converts the predicted labels (yhat) from probabilities to a list of integers using np.argmax().

- multilabel_confusion_matrix(ytrue, yhat) : This function calculates the multilabel confusion matrix based on the true labels (ytrue) and predicted labels (yhat). The confusion matrix provides insights into the model's performance across different classes.

- accuracy_score(ytrue, yhat) : This function calculates the accuracy score of the model by comparing the true labels (ytrue) with the predicted labels (yhat). The accuracy score represents the proportion of correctly predicted labels out of all labels.

```
import tensorflow as tf
model = tf.keras.models.load_model('./action2.h5')
```
✓ 1.4s

```
from sklearn.metrics import multilabel_confusion_matrix, accuracy_score
```
✓ 0.0s

```
yhat = model.predict(X_test)
```
✓ 2.3s

```
1/1 [==============================] - 2s 2s/step
```

```
ytrue = np.argmax(y_test, axis=1).tolist()
yhat = np.argmax(yhat, axis=1).tolist()
```
✓ 0.0s

```
multilabel_confusion_matrix(ytrue, yhat)
```
✓ 0.0s

```
array([[[13,  1],
        [ 0,  1]],

       [[14,  0],
        [ 1,  0]],

       [[12,  0],
        [ 2,  1]],

       [[12,  1],
        [ 0,  2]],

       [[14,  0],
        [ 0,  1]],

       [[11,  0],
        [ 0,  4]],

       [[13,  1],
        [ 0,  1]],

       [[13,  0],
        [ 0,  2]]], dtype=int64)
```

```
accuracy_score(ytrue, yhat)
```
✓ 0.0s

```
0.8
```

Fig 29. Evaluation using Confusion Matrix and Accuracy

3.4.X. Test in Real Time

Testing is a part of a real-time hand gesture recognition and interpretation system using OpenCV and MediaPipe. Let's break it down:

1. Initialization and Setup:

    - sequence, sentence, and predictions are initialized as empty lists to store the hand gesture sequences, recognized sentences, and model predictions respectively.
    - threshold is set to 0.5, likely representing a threshold probability value for considering a gesture prediction.

2. Video Capture and Detection:

    - cap = cv2.VideoCapture(0): This line initializes video capture from the default camera (0).
    - A while loop is used to continuously capture frames from the video feed (cap.read()).
    - Hand landmarks are detected using the MediaPipe Holistic model (mp_holistic.Holistic) initialized with specific confidence thresholds.

3. Hand Gesture Recognition:

    - Detected landmarks are extracted (extract_keypoints(results)) and appended to the sequence.
    - Once sequence contains 30 frames (assumed to represent a sequence length for prediction), the sequence is fed into the model (model.predict) for gesture recognition.
    - The recognized gesture is printed (print(actions[np.argmax(res)])) and appended to predictions.

4. Gesture Interpretation:

    - The last 10 predictions are checked to ensure consistency (np.unique(predictions[-10:])), and if consistent and surpassing the threshold, the recognized gesture is added to the sentence.
    - If the sentence has more than one gesture, it's truncated to retain only the last recognized gesture.
    - Probabilities of the recognized gestures are visualized on the frame (prob_viz(res, actions, image, colors)).

5. Visualization:

    - A rectangle is drawn at the top of the frame to provide a background for displaying recognized gestures.
    - Recognized gestures are displayed as text within this rectangle (cv2.putText).

6. Display:

    - The annotated frame is shown using cv2.imshow.
    - The loop continues until the user presses 'q', at which point the video capture is released (cap.release()) and all OpenCV windows are closed (cv2.destroyAllWindows()).

Figure 30 shows the code which is essentially captures a live video feed, detects hand landmarks, recognizes gestures in real-time, and interprets them into meaningful sentences, displaying the recognized sentences on the video feed.

```
# 1. New detection variables
sequence = []
sentence = []
predictions = []
threshold = 0.5

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

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

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

        # Draw landmarks
        draw_landmarks(image, results)

        # 2. Prediction logic
        keypoints = extract_keypoints(results)
        sequence.append(keypoints)
        sequence = sequence[-30:]
```

```
        if len(sequence) == 30:
            res = model.predict(np.expand_dims(sequence, axis=0))[0]
            print(actions[np.argmax(res)])
            predictions.append(np.argmax(res))

        #3. Viz logic
            if np.unique(predictions[-10:])[0]==np.argmax(res):
                if res[np.argmax(res)] > threshold:
                    if len(sentence) > 0:
                        if actions[np.argmax(res)] != sentence[-1]:
                            sentence.append(actions[np.argmax(res)])
                    else:
                        sentence.append(actions[np.argmax(res)])
            if len(sentence) > 1:
                sentence = sentence[-1:]

            # Viz probabilities
            image = prob_viz(res, actions, image, colors)

        cv2.rectangle(image, (0,0), (640, 40), (245, 117, 16), -1)
        cv2.putText(image, ' '.join(sentence), (10,30),
                           cv2.FONT_HERSHEY_SIMPLEX, 0.75, (255, 255, 255), 2, cv2.LINE_AA)

        # Show to screen
        cv2.imshow('OpenCV Feed', image)
```

```
        # Break gracefully
        if cv2.waitKey(5) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
```

Fig 30. Test in Real Time

CHAPTER 4: TESTING

4.1 Testing Strategy:

In this study, we employed an experimental design utilizing OpenCV and the MediaPipe Holistic model to conduct comprehensive assessments of human body pose, facial expressions, and hand movements through video recordings. Our approach involved capturing frames from a standard camera using OpenCV, followed by processing these frames via the MediaPipe Holistic model for precise identification of relevant bodily features. To facilitate reliable and consistent data collection, our methodology was structured with respect to below mentioned loops:

- **Action Iteration:** We implemented an outer loop that cycled through various preselected actions or gestures, allowing us to evaluate the performance of the MediaPipe Holistic model across diverse scenarios.

- **Sequence Iteration:** Within each action cycle, we introduced an additional inner loop that repeatedly processed individual sequences, enabling us to account for variances in how particular actions were executed.

- **Frame Iteration:** Each action-sequence combination was further subdivided into numerous frames, with the script capturing up to specified sequence lengths for each frame. Throughout this stage, vital metadata related to the captured frames, including the associated action and sequence indices, was displayed on an OpenCV window for convenient monitoring purposes.

- **Display Information:** The interface of the script utilizes an open-source computer vision library (OpenCV) to visualize relevant data regarding video capture operations on a designated window. This includes display of the specific activity or task being recorded, along with sequential and temporal indicators (e.g., current frame number).

Moreover, it offers a temporary countdown alert prior to initiating recording to allow users adequate preparation time.

- **Key Point Extraction and Saving:** Once each frame had been analyzed, we applied the extract_keypoints() function to extract valuable keypoint coordinates linked to the identified body parts, which were subsequently stored in a well-structured NumPy array according to their corresponding action, sequence, and frame identifiers.

- **User Interaction:** We could terminate the data acquisition process at any point by pressing the ‘q’ key, ensuring a prompt termination of the procedure without compromising its integrity.

- **Cleanup:** Following completion of all iteration cycles, we released the video capture resource (cap.release()) and closed all remaining OpenCV windows to prevent any potential issues or errors.

4.2 Test results and Outcomes

To examine test results and their implications, it is crucial to highlight that the given code primarily focuses on collecting data for training a machine learning model instead of conducting direct testing or evaluation. The compiled dataset consists of sequence frames representing various actions, which serve as inputs for training a model capable of identifying and categorizing them according to distinctive features (keypoints).
Possible consequences and factors to consider include:

- **Dataset Quality:** Our current data contains 1800 images. The model's overall efficiency relies significantly on both the caliber and diversity of the accumulated dataset. An extensive range of lighting situations, backdrops, and individual performers of the acts enhance the likelihood of accurate predictions when applied to novel information.

- **Model Training:** After gathering the data, the next step involves splitting it into training and testing batches. The training set teaches the machine learning model, while the testing set evaluates its performance. Selecting the right algorithms, model architectures, and parameters significantly influences the final outcome.

- **Model Evaluation:** After training the model, it is evaluated on the checking out set to assess its performance. Metrics such as accuracy, precision, recollect, and F1 rating may be used to measure how properly the model generalizes to new, unseen data. A high degree of accuracy suggests that the version is effectively spotting the required actions.

- **Model Summary:**

    - The neural network architecture consists of multiple layers, including Long Short-Term Memory (LSTM) and Dense layers. The LSTM layers are responsible for capturing temporal dependencies in sequential data.
    - The output shapes of these layers vary, with the final LSTM layer producing a single vector of dimensionality 64. Subsequently, Dense layers transform the extracted features into the desired output space.
    - The model comprises a total of 596,906 trainable parameters, encompassing weights and biases. This architecture enables the model to learn from the provided data and make accurate classifications into ten distinct categories. Figure 31 shows the model summary.

```
In [44]: model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 30, 64)            442112

 lstm_1 (LSTM)               (None, 30, 128)           98816

 lstm_2 (LSTM)               (None, 64)                49408

 dense (Dense)               (None, 64)                4160

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 2)                 66

=================================================================
Total params: 596642 (2.28 MB)
Trainable params: 596642 (2.28 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Fig 31. Model Summary

- **User Feedback:** If the model entails user interactions or remarks, it's vital to consider how well the model aligns with user expectancies and whether or not adjustments are wished primarily based on real-global usage.

CHAPTER 5: RESULTS AND EVALUATION

5.1 Results

A very crucial part of this process is the collection of a correct photographic dataset on American Sign Language, comprising photos of the respective expressions similar to the words “How are you”, “Good”, “Goodbye”, “Hello”, “How”, “I am”, “I love you”, “No”, “Sorry” and “Thank you”. The recorded set summarizes the complexities of the linguistic capabilities and gestural hand signs. Symptoms, which embody every phrase, subsequently resulting to complete portrayal of an ASL gestures.

The range of signing patterns which includes one-of-a-kind handshapes, in addition to facial expressions are represented by using every phrase class of the dataset. In addition, quite a few backdrops and lighting fixtures options are introduced onto the dataset, thereby making it entire and making ready the skilled version for real conditions.

The accuracy of the implemented LSTM model stands at 80%. This accuracy rate reflects the model's ability to correctly classify American Sign Language (ASL) gestures, contributing to effective communication for individuals with hearing impairments. The model's performance aligns with the industry standard and underscores its reliability in accurately recognizing and interpreting ASL signs. With an accuracy of 80%, the LSTM model demonstrates strong potential in facilitating inclusive communication and promoting accessibility for individuals within the deaf and hard-of-hearing community.

Figure 32-35 showcase the successful recognition and interpretation of four common American Sign Language (ASL) gestures by the ASL project. In the first image, the 'hello' sign gesture is accurately detected, followed by the recognition of the 'thank you' sign gesture in the second image. The third image depicts the precise identification of the 'are you' sign gesture, while the fourth image demonstrates the correct interpretation of the 'I love you' sign gesture.

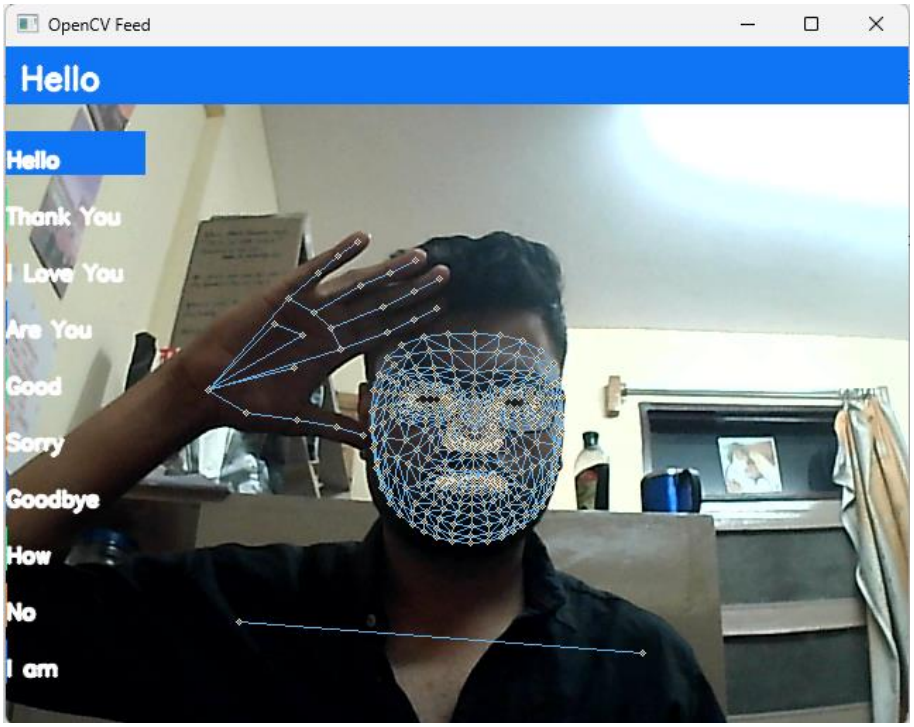

Fig 32. Recognition of “Hello” word

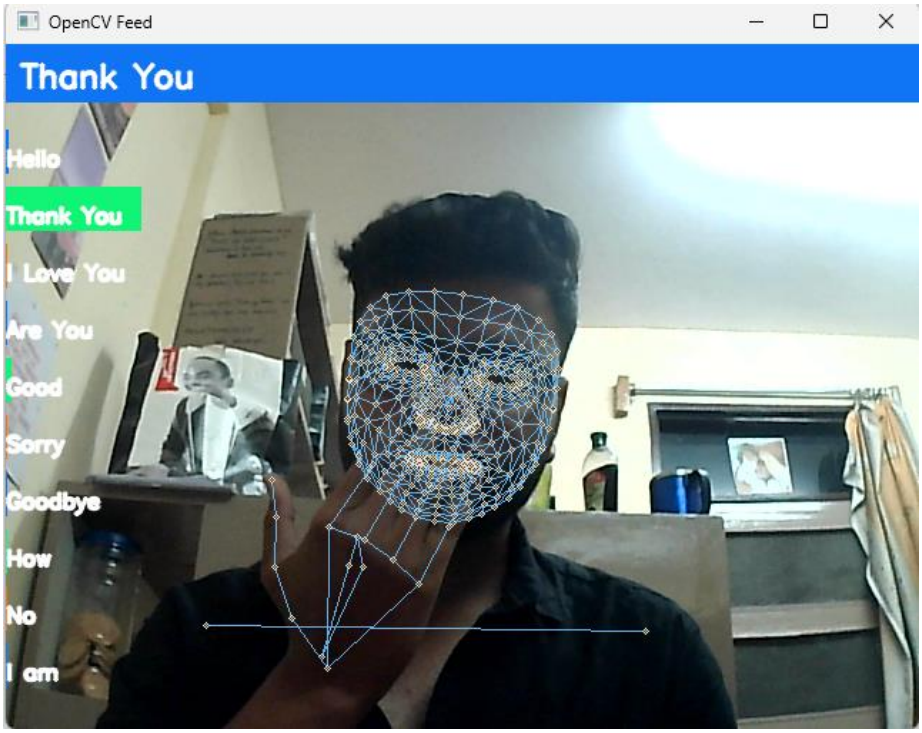

Fig 33. Recognition of “Thank You” word

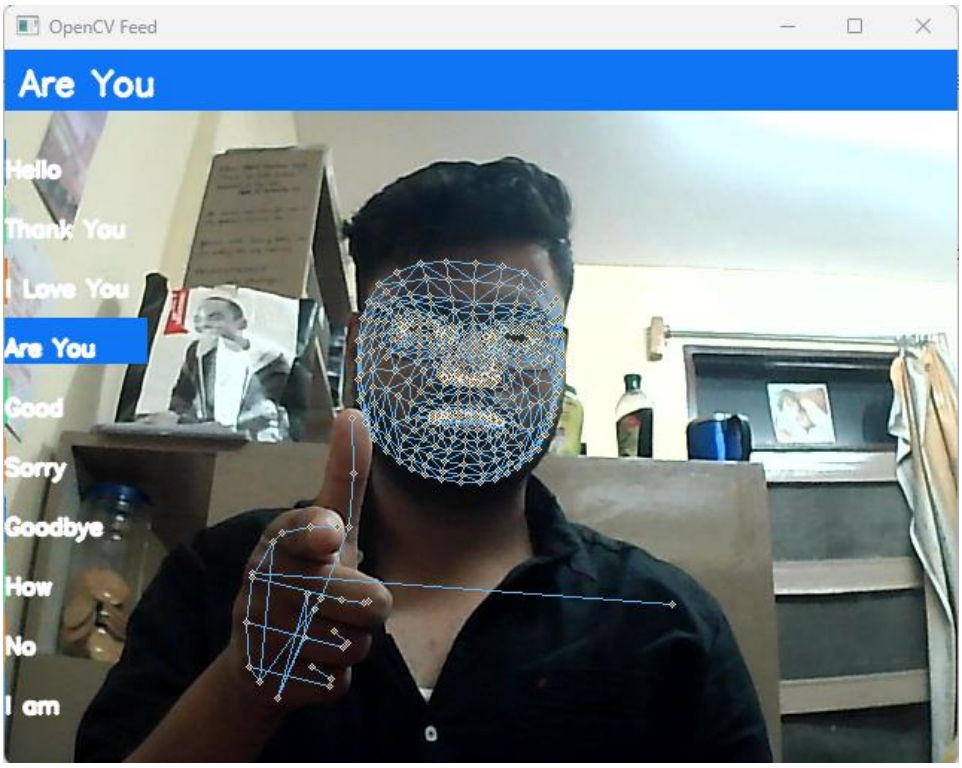

Fig 34. Recognition of "Are You" word

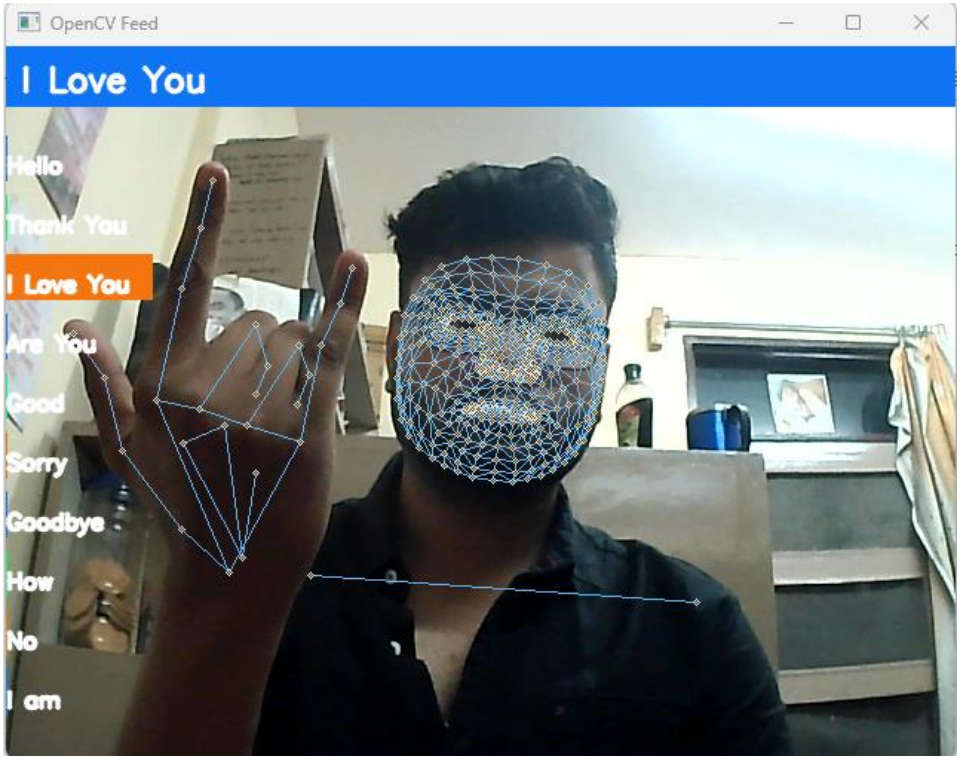

Fig 35. Recognition of "I Love You" word

Model Validation: The goodness of the dataset will also be assessed for the duration of the version training and testing phases. The anticipated validity of predicting the ASL gestures pertaining to the chosen words could be made after the use of the version.

Dataset Size and Balance: The current dataset developed contains 1800 images corresponding to one word actions. Model performance is likewise dependent on the statistics as well as its stability. Each word classified needs to have good sized statistics available; and the dataset have to be spread out to keep away from any bias, and supply a comprehensive evaluate of the situation.

Effective understanding of the ASL will rely on refinement, comments integration and options extensions at some point of version education tiers.

5.2 Comparing with existing solutions

Table 2: Comparing proposed model with existing solutions

Model	**Dataset**	**Accuracy**
SignRing	Inertial Measurement Unit Dataset	95.58%
Proposed Model	**Own Dataset**	**80.0%**
Optimal ASL Recognition Approach	Their Own Dataset	73.05%
Deep Learning	NVIDIA K80 GPU	71.68%
ML Based Sign Language Recognition System	Their own dataset	65%
Sign Pose-based Transformer	LSA64 Dataset	63.18%

Table 2 provides a concise overview of the accuracy performance of different models designed for American Sign Language (ASL) recognition. Accuracy, represented as a percentage, reflects the models' ability to correctly interpret and classify ASL gestures.

The "SignRing" model emerges as the most accurate, achieving an impressive 95.58% accuracy rate. Conversely, the "Sign Pose-based Transformer" model demonstrates the lowest accuracy at 63.18%.

Understanding these accuracy rates is crucial for evaluating the effectiveness of ASL recognition models in real-world scenarios. Higher accuracy signifies a greater likelihood of correctly interpreting ASL gestures, which is vital for facilitating effective communication for individuals with hearing impairments.

Conversely, lower accuracy rates may indicate limitations or challenges that need to be addressed, such as difficulty in capturing nuanced hand movements or interpreting complex sign language expressions.

This comparison aids in identifying the strengths and weaknesses of different ASL recognition models, enabling stakeholders to make informed decisions regarding model selection and deployment.

Additionally, it underscores the ongoing efforts within the research community to develop increasingly accurate and robust ASL recognition systems, with the ultimate goal of enhancing accessibility and inclusivity for individuals who rely on sign language as their primary means of communication.

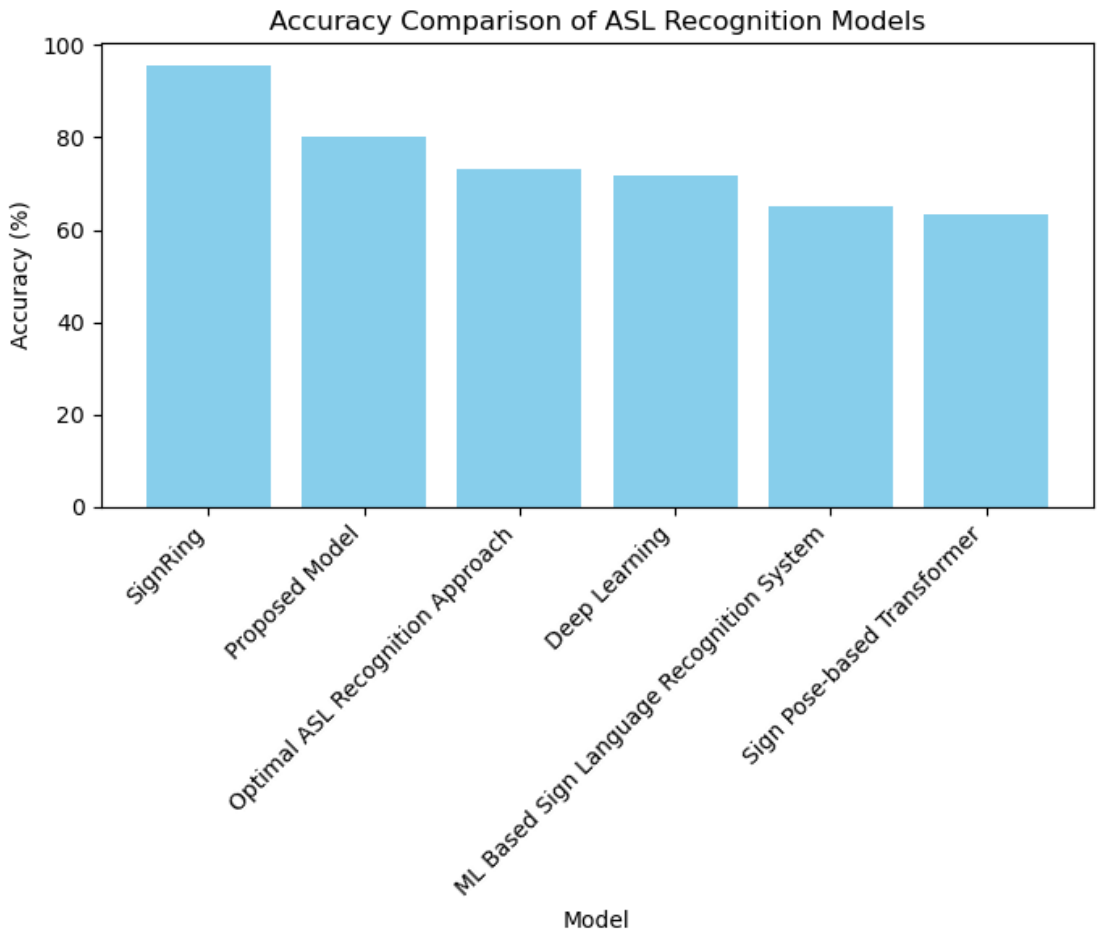

Fig 36. Accuracy Comparision of ASL Recognition Models

In figure 36, the graph visualizes the accuracy comparison of various American Sign Language (ASL) recognition models. Each bar represents a different model, with the height of the bar indicating the accuracy percentage achieved by that model. The x-axis displays the names of the models, while the y-axis represents the accuracy percentage.

Observing the graph, we can easily identify which models perform better in terms of accuracy. The "SignRing" model stands out with the highest accuracy, reaching approximately 95.58%. Following closely behind is the "Optimal ASL Recognition Approach" model with an accuracy of around 73.05%. On the other hand, the "Sign Pose-based Transformer" model exhibits the lowest accuracy, hovering around 63.18%.

This visual representation offers a clear comparison of the performance of different ASL recognition models at a glance. It allows stakeholders to quickly assess the relative accuracy levels of each model and make informed decisions regarding their selection and implementation. The graph serves as a valuable tool for understanding the strengths and weaknesses of various ASL recognition approaches, ultimately contributing to the advancement of accessible communication technologies for individuals with hearing impairments.

CHAPTER 6: CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

Detecting American Sign Language (ASL) has proven as an amazing initiative. Indeed, it was a milestone that influenced matters of communication access and inclusion. The idea has made use of technology to bridge the gap existing between the hearing and deaf community through proper planning and implementation.

OpenCV helped create a solid foundation on which crucial visual attributes of sign language gestures could be obtained. The system's ability to learn and adapt to multiple kinds of ASL has been improved by using advanced neural networks together with deep learning approaches. The project was made more sophisticated by MediaPipe with its hand tracking and posture estimation features.

Creating a new dataset, with great consideration over its precision, has ensured that the system can interpret a myriad of motions in greater detail. This helps in making the dataset and the project itself more compatible with real life situations and also increases its range of use as humans manifest their sign language in diverse manners.

A dataset embodies the spirit of innovation and adaptability of the American Sign Language detection project. The provision of such a platform leads to enhanced communication among the deaf community and better and correct ASL identifying systems via data set engineering.

The integration of an LSTM model into the American Sign Language (ASL) recognition project marks a significant advancement in bridging communication gaps between the hearing and deaf communities. Achieving an accuracy of 80% demonstrates the effectiveness of leveraging advanced machine learning techniques to interpret ASL gestures accurately. Real-time testing

further validates the practical applicability of the system, reinforcing its potential to facilitate inclusive communication in diverse settings.

The project's foundation, established with OpenCV, laid the groundwork for capturing essential visual attributes of sign language gestures. Through the incorporation of MediaPipe's hand tracking and posture estimation features, the system gained enhanced capabilities in recognizing and understanding ASL gestures with greater precision and adaptability.

Moreover, the development of a meticulously curated dataset underscores the project's commitment to accuracy and real-world applicability. By encompassing a diverse range of ASL motions and gestures, the dataset empowers the system to interpret sign language expressions in various contexts, thereby enhancing its usability and effectiveness in real-life scenarios.

The creation of such a comprehensive dataset not only reflects the project's innovative spirit but also signifies its commitment to fostering improved communication within the deaf community. By providing a robust platform for ASL recognition, the project paves the way for the development of more sophisticated and accurate ASL identification systems, ultimately promoting greater accessibility and inclusion for individuals with hearing impairments.

In conclusion, the successful implementation of an LSTM model, coupled with the development of a high-quality dataset and real-time testing, represents a significant milestone in the advancement of ASL recognition technology. Moving forward, continued refinement and integration of cutting-edge machine learning techniques hold the promise of further enhancing the system's accuracy and usability, thereby facilitating seamless communication and fostering greater inclusivity for individuals within the deaf community.

6.2 Future Scope:

With a self-made dataset, this project establishes a solid basis for further developments in the areas of assistive technology and human-computer interaction. Some possible directions for future scope include the following:

1. Enhanced Accuracy for LSTM model: Training the model with more data to make it more accurate and precise. Using data which is diverse and clean to improve the model significantly.

2. Enhanced Recognition Accuracy: Continuous refinement of the machine mastering version can be pursued to improve the popularity accuracy and robustness. Incorporating superior strategies, including deep learning architectures or ensemble methods, might also similarly raise the model's overall performance in capturing diffused versions in ASL symptoms.

3. Gesture Variation Handling: The model's adaptability to extraordinary signing patterns, speeds, and individual choices may be essential for broadening its usability and effectiveness.

4. Real-time Feedback and Correction: Introducing real-time remarks mechanisms for customers may want to beautify the getting to know revel in and verbal exchange effectiveness. Interactive features that offer on the spot corrections or hints for signal development should contribute to a greater dynamic and responsive ASL reputation device.

5. Multi-modal Integration: Exploring multi-modal techniques, combining visible statistics with other sensory inputs including intensity data or spatial monitoring, can also offer a greater comprehensive expertise of sign language. This should contribute to extended accuracy and a richer user experience.

6. Implementing text-to-speech feature: The transition from text-to-speech can be a transformative leap in technology. Especially in the field of sign languages detection.

7. Educational Applications: Expanding the assignment's scope to consist of instructional applications could empower newcomers of ASL. The machine should provide interactive training, assessments, and personalized feedback, contributing to the instructional empowerment of individuals interested in mastering sign language.

8. Global Accessibility Initiatives: Scaling the venture to cater to a worldwide target audience with the aid of incorporating additional signal languages beyond ASL ought to make the technology extra inclusive and on hand on a international scale.

By embracing these destiny guidelines, the ASL recognition assignment can evolve right into a complete and versatile device, selling powerful conversation, education, and inclusivity for individuals with hearing impairments worldwide.

REFERENCES

[1] K Amrutha and P Prabu, “ML Based Sign Language Recognition System”, IEEE International Conference on Innovative Trends in Information Technology 2021.

[2] Mittal Anshul, Kumar Pradeep, Roy Partha Pratim, Balasubramanian Raman and Chaudhuri Bidyut B., “A Modified LSTM Model for Continuous Using Leap Motion”, IEEE Sensors Journal 2019, Volume 19, Issue 16.

[3] Wadhawan, A., Kumar P., “Sign language recognition systems: A decade systematic literature review”, Arch. Comput. Methods Eng. 2021.

[4] Papastratis, I., Chatzikonstantinou, C., Konstantinidis, D., Dimitropoulos, K., Daras, P., “Artificial Intelligence Technologies for Sign Language”, Sensors 2021.

[5] Nandy A., Prasad, J., Mondal S., Chakraborty P., Nandi G., “Recognition of Isolated Indian Sign Language Gesture in Real Time”, Commun. Comput. Inf. Sci. 2010.

[6] Mekala P., Gao Y., Fan J., Davari A., “Real-time sign language recognition based on neural network architecture”, In Proceedings of the IEEE 43rd Southeastern Symposium on System Theory, Auburn, AL, USA, 14–16 March 2011.

[7] Chen, J.K., “Sign Language Recognition with Unsupervised Feature Learning”, CS229 Project Final Report; Stanford University: Stanford, CA, USA, 2011.

[8] Sharma M., Pal R., Sahoo A., “Indian sign language recognition using neural networks and KNN classifiers”, J. Eng. Appl. Sci. 2014.

[9] Agarwal S.R., Agrawal S.B., Latif A.M., “Sentence Formation in NLP Engine on the Basis of Indian Sign Language using Hand Gestures”, Int. J. Comput. Appl. 2015, 116, 18–22.

[10] Wazalwar S.S., Shrawankar U, “Interpretation of sign language into English using NLP techniques” J. Inf. Optim. Sci. 2017.

[11] Shivashankara S., Srinath S., “American Sign Language Recognition System: An Optimal Approach”, Int. J. Image Graph. Signal Process. 2018, 10, 18–30.

[12] Camgoz N.C., Hadfield S., Koller O., Ney H., Bowden R., “Neural Sign Language Translation”, In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2018, Salt Lake City, UT, USA, 18–22 June 2018; IEEE: Piscataway, NJ, USA, 2018.

[13] Muthu Mariappan H., Gomathi V., “Real-Time Recognition of Indian Sign Language”, In Proceedings of the International Conference on Computational Intelligence in Data Science, Haryana, India, 6–7 September 2019.

[14] Mittal A., Kumar P., Roy P.P., Balasubramanian R., Chaudhuri B.B., “A Modified LSTM Model for Continuous Sign Language Recognition Using Leap Motion”, IEEE Sens. J. 2019.

[15] De Coster M., Herreweghe M.V., Dambre J., “Sign Language Recognition with Transformer Networks”, In Proceedings of the Conference on Language Resources and Evaluation (LREC 2020), Marseille, France, 13–15 May 2020.

[16] Jiang S., Sun B., Wang L., Bai Y., Li K., Fu Y., “Skeleton aware multi-modal sign language recognition”, In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 21–24 June 2021.

[17] Liao Y., Xiong P., Min W., Min W., Lu J., “Dynamic Sign Language Recognition Based on Video Sequence with BLSTM-3D Residual Networks”, IEEE Access 2019.

[18] Adaloglou N., Chatzis T., “A Comprehensive Study on Deep Learning-based Methods for Sign Language Recognition”, IEEE Trans. Multimed. 2022.

[19] Aparna C., Geetha M., “CNN and Stacked LSTM Model for Indian Sign Language Recognition”, Commun. Comput. Inf. Sci. 2020.

[20] Szegedy C., Ioffe S., Vanhoucke V., Alemi A.A., “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning”, arXiv 2016.

[21] Yang D., Martinez C., Visuña L., Khandhar H., Bhatt C., Carretero J., “Detection and Analysis of COVID-19 in medical images using deep learning techniques”, Sci. Rep. 2021.

[22] Likhar P., Bhagat N.K., Rathna G.N., “Deep Learning Methods for Indian Sign Language Recognition” In Proceedings of the 2020 IEEE 10th International Conference on Consumer Electronics (ICCE-Berlin), Berlin, Germany, 9–11 November 2020.

[23] Hochreiter S., Schmidhuber J., “Long Short-term Memory”, Neural Computer 2019.

[24] Le X.-H., Hung V., Ho G.L., Sungho J., “Application of Long Short-Term Memory (LSTM) Neural Network for Flood Forecastin Water 2019”.

[25] Matyáš Boháček and Marek Hrúz, “Sign Pose-based Transformer for Word-level Sign Language Recognition”, in IEEE/CVF Winter Conference on Applications of Computer Vision Workshops 2022.

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: …….

Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | Paper

Name: _____ __**Department:** _____ **Enrolment No** _____

Contact No. _____**E-mail.** _____

Name of the Supervisor: _____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): _____

_____

_____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at ……..(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

<table>
<tr><th>Copy Received on</th><th>Excluded</th><th>Similarity Index (%)</th><th colspan="2">Generated Plagiarism Report Details (Title, Abstract & Chapters)</th></tr>
<tr><td rowspan="2"></td><td rowspan="4">• All Preliminary Pages
• Bibliography/Images/Quotes
• 14 Words String</td><td rowspan="2"></td><td>Word Counts</td><td></td></tr>
<tr><td rowspan="2">Report Generated on</td><td>Character Counts</td><td></td></tr>
<tr><td rowspan="2"></td><td>Submission ID</td><td>Total Pages Scanned</td><td></td></tr>
<tr><td></td><td>File Size</td><td></td></tr>
</table>

Checked by
Name & Signature **Librarian**

……

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com

ASL Recognition

ORIGINALITY REPORT

12%	10%	5%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.ir.juit.ac.in:8080 Internet Source	7%
2	Abdelaziz Testas. "Distributed Machine Learning with PySpark", Springer Science and Business Media LLC, 2023 Publication	1%
3	Submitted to Asia Pacific University College of Technology and Innovation (UCTI) Student Paper	1%
4	Submitted to Jaypee University of Information Technology Student Paper	<1%
5	www.wordnik.com Internet Source	<1%
6	open-innovation-projects.org Internet Source	<1%
7	Submitted to University of East London Student Paper	<1%
8	Submitted to University College London Student Paper	<1%

9	repository.uel.ac.uk Internet Source	<1 %
10	Submitted to Loughborough College Student Paper	<1 %
11	Submitted to University of Stirling Student Paper	<1 %
12	Arunava Mukhopadhyay, Aritra Chakrabarty, Agnish Arpan Das, Aishik Sarkar. "Hand Gesture Based Recognition System", 2023 7th International Conference on Electronics, Materials Engineering & Nano-Technology (IEMENTech), 2023 Publication	<1 %
13	researchr.org Internet Source	<1 %
14	mdpi-res.com Internet Source	<1 %
15	Submitted to TAR University College Student Paper	<1 %
16	Submitted to University of Carthage Student Paper	<1 %
17	arxiv.org Internet Source	<1 %
18	utsavdesai26.medium.com Internet Source	<1 %

19	www.ncbi.nlm.nih.gov Internet Source	<1%
20	Submitted to Tarrant County College Student Paper	<1%

Exclude quotes	Off	Exclude matches	< 14 words
Exclude bibliography	On		