University of Bahrain

College of Information Technology

Department of Computer Engineering

SIGN LANGUAGE GESTURE RECOGNITION USING COMPUTER VISION IN REAL-TIME

Prepared by

Puja Karmakar 20194981

Afifa Mohamed Ismaeel Altorabi 20183087

Zainab Redha Ebrahim Matrook 20195982

For

ITCE 499/ ITNE 402

Senior Project

Academic Year 2022-2023-Semester 2

Project Supervisor: Dr. Jalal Mohamed Mufleh Khlaifat

Date of Submission: 17-May-2023

Abstract

Sign Language Gesture Recognition (SLGR), with its ability to decipher hand and body movements into meaningful signs, serves as a bridge of communication between the deaf and the hearing. By unlocking the language of sign, this technology empowers individuals who face hearing impairments to express themselves freely and connect with others on a deeper level. In our project, we propose a Machine Learning (ML) model for SLGR using computer vision in real time. The proposed model employs a deep learning-based approach, specifically a Convolutional Neural Network (CNN) model Long Short-Term Model (LSTM) and Support Vector Model (SVM), to recognize and classify various sign language gestures in real time. The ML models were trained on a 45 Word-Level dataset of Signer's gesture videos, which were preprocessed and segmented into individual frames using OpenCV and Media Pipe Feature Engineering. The proposed model was tested on a real-time video stream, achieving an accuracy of 95% for American Sign Language (ASL) gestures. The real-time performance of the proposed model was evaluated using various frame rate, processing time, classification and prediction. The results show that our ML model can recognize sign language gestures in real time with high accuracy and efficiency. It can be used as an educational tool for individuals who wish to learn sign language, thereby increasing social inclusion and promoting diversity.

Acknowledgments

Firstly, we would like to thank our project supervisor Dr. Jalal Mohamed Mufleh Khalifat, for his invaluable guidance, support, and encouragement throughout the project. His expertise, advice, and feedback have been instrumental in shaping our project and ensuring its successful completion.

We would also like to thank Dr. Mohamed A. Almeer, Dr. Alauddin Yousif Al-Omary, and Ahmed Alsaigh, who provided us with their technical expertise and support throughout the project. Their valuable insights, suggestions, and assistance were crucial in overcoming technical challenges and achieving our project goals.

We are grateful to our friends and family for their unwavering support and encouragement throughout our academic journey. Their invaluable assistance in proofreading and providing feedback on this research has refined and enhanced its quality. We thank them for their love, motivation, and encouragement that have been a constant source of inspiration for us. We are truly grateful for their unwavering support and encouragement throughout this journey.

Finally, we thank the University's Computer Engineering Department faculty members and staff, who have provided us with a nurturing academic environment, excellent facilities, and resources essential in completing this project.

We are grateful for the opportunity to work on this project and the support we received from everyone involved.

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

According to a study by World Health Organization (WHO) reports that by 2050 2.5 billion people, that is 1 in every 10 people, are projected to have some degree of hearing loss and at least 700 million will require hearing rehabilitation. Including over 1 billion young adults are at risk of permanent, avoidable hearing loss due to unsafe listening practices [1].

Deaf people are affected by many forms of exclusion, especially now in the pandemic world. Our project aims to build a Deep Learning solution to make the world more accessible for the Deaf community and increase the existing knowledge base in using Deep Learning for American Sign Language (ASL). Sign languages are the primary means of communication used by the deaf community and possess distinctive linguistic properties. Sign language interpretation methods aim to automatically translate sign languages using visual techniques. This process involves two primary tasks, namely, recognizing individual signs (referred to as "word-level" or "isolated sign language recognition") and recognizing sign language sentences (referred to as "sentence-level" or "continuous sign language recognition").

Our project focuses on the word-level recognition task of American Sign Language (ASL), which is widely used by deaf communities across more than 20 countries worldwide [2].

1.1 Problem Statement

Speech and hearing-impaired people use hand signs to communicate, hence normal people face problems in recognizing their language. Hence there is a need for systems which recognize the different signs and convey the information to normal people. Our proposed system will help in breaking down these barriers for Sign Language (SL) by providing interactive integrated recognition capabilities. The system will achieve significant improvements in SL recognition and establish stability and simplicity in communication with Speech and hearing-impaired and ordinary people communication. The system will enable improved hybrid recognition using visual and depth information from RGB-D cameras.

The proposed system will integrate deep learning-based image and video recognition and visual concept detection with continuous sign language recognition. The system will provide a common platform for seamless interaction and understanding between hearing-impaired and normal speech communication abilities. Our system aims to develop intelligent interfaces to facilitate improved communication by combining sign language recognition with visual concept detection and natural language generation technologies. The system will help in automatic generation of text/speech translations for enhancing accessibility and inclusiveness.

1.2 Project Objectives

The primary goal of this project is to develop real-time Sign Language Action Gesture Detection using Computer Vision techniques. To achieve this goal, the following objectives have been defined to guide the project's development:

1. Investigate Machine Learning (ML) models that evaluate different feature extraction using MediaPipe and classification methods to enhance the precision and dependability of the gesture detection model.
2. Optimize the performance of each model with respect to speed and computational efficiency, to ensure real-time operation.
3. Train the models with various numbers of sequences, frames, and words to assess how these variables affect model accuracy.
4. Demonstrate the effectiveness of the models through real-time recognition of sign language gestures captured by some different cameras.
5. Compare the findings to determine which model performed the best accuracy after rigorously testing and evaluating each one.
6. Document the design and implementation of each model, including the hardware and software components utilized on the Raspberry Pi3.
7. Provide a critical appraisal of each model's strengths and limits, and areas for development and future work.
8. Encourage better accessibility and inclusivity in society by contributing to the development of computer vision and sign language recognition.

1.3 Relevance/Significance of the project

The SLGR project's significance lies in its potential to improve the quality of life for individuals with hearing impairments by providing them with a more efficient and accurate means of communication. The proposed model framework's real-time performance ensures that communication is not delayed, enabling individuals with hearing impairments to participate in real-time conversations. It can also be used to develop new and innovative assistive technologies for individuals with hearing impairments. For example, the model can be used to develop smart home systems that recognize sign language gestures to control various devices and appliances. Also, it can also be used in various healthcare settings, such as hospitals and clinics, to improve communication between healthcare providers and patients with hearing impairments. This can improve the quality of care and patient outcomes by ensuring that patients can communicate their needs and concerns effectively. Additionally, the project can be used as an educational tool for individuals who wish to learn sign language, thereby increasing social inclusion and promoting diversity.

1.4 Project Methodology

1. **Data collection:** The initial phase of the proposed system is to collect data. This involves defining a path for exported data by using NumPy arrays, which consist of 45 words and 30 frames for each phrase, by utilizing a webcam using OpenCV to record videos of Signers doing various SL gestures. Subsequently, we will extract the key points and export them on a NumPy array to make our unique dataset.

2. **Data preprocessing:** A preprocessing step is required after the data has been collected to generate a labeled map which is a representation of all the elements within the mode. It includes the label for each signed word. The label map has 45 labels, each of which represents a distinct word. Each label has a distinct id ranging from 0 to 44.

3. **Feature extraction:** The following step is to extract crucial features from our data by minimizing the number of features in our dataset by constructing derived features from the existing ones and then eliminating the original features. This requires recognizing critical components of hand landmarks, hand motions, and gesture postures using computer vision algorithms.

4. **Model training:** Following the extraction of the characteristics, a machine learning model is trained by utilizing the TensorFlow object detection API to recognize the various sign language motions by dividing the data into 70% for training and 20% for testing.

5. **Real-time detection:** Once the features have been extracted, a machine-learning model is trained to detect the various sign language gestures using cv2 and NumPy dependencies. Our system identifies indications in real-time by drawing landmarks using a mediapipe model and translating what each motion signifies into English.

6. **Output generation:** In the final phase, the detected sign language gestures are translated into written words using natural language processing, providing real-time communication.

1.5 Report Outline

This chapter demonstrates an introduction to the project by discussing the problem statement, project objectives, project relevance/significance, and project scope and limitations. The rest of the report will discuss these topics as follows:

- **Chapter 2 (Background and Literature Review)** summarizes previous research articles regarding Hand gesture identification, machine learning models, and computer vision implications.
- **Chapter 3 (System Design)** describes the project's techniques and the strategy used in Data Collection, Features Extraction, ML model building, evaluation and training.
- **Chapter 4 (System Implementation)** represents the software implementation and Raspberry Pi implications inclosing topics like packages, tools, platforms, hardware components used to implement our system. In addition to challenges and outcome.
- **Chapter 5 (Testing and Results)** provided screenshots of the tested algorithms and the results of ML training models in terms of testing accuracy, performance and Predictions.
- **Chapter 6 (Conclusion and Future Work)** presents an overview of the project's implementation process and some enhancements regarding the proposed algorithm deployment and recommendations for future improvements to the proposed system.

Chapter 2
Background and Literature Review

Sign Language Gesture Recognition technology is an emerging field that has gained significant attention in recent years due to the potential it holds for improving communication access and quality of life for the deaf and hard of hearing community. The increasing prevalence of digital communication platforms and the need for accessibility have driven researchers and developers to explore the possibilities of this technology.

Several studies have investigated the potential of Sign Language Gesture Recognition technology to enhance the accessibility of healthcare, education, and other essential services for the deaf and hard of hearing community. Researchers have explored the use of wearable sensors and cameras to detect and interpret sign language gestures, as well as the use of artificial intelligence and machine learning algorithms to improve the accuracy and speed of recognition.

Hand gesture identification is a challenging topic to solve in machine learning. Murakami and Taguchi released the first research publication in sign language recognition using a neural network in 1991 [3]. With advancements in the field of computer vision, several researchers have devised unique techniques to assist the physically challenged community. Consequently, some earlier research actively separate hands and/or faces before extracting features; they utilize colored gloves or data gloves to track hand motions and more correctly cope with segmentation and occlusion concerns [4]. Wang and Popovic created a real-time hand-tracking program using colored gloves [5]. The color pattern of the gloves was recognized by the K-Nearest Neighbors (KNN) technique, but the system requires continuous feeding of hand streams, and the requirement of wearing gloves always is impractical in daily life, and data gloves with probes frequently limit the natural movements of the signers. Recognizing hand gestures is a difficult topic to address in machine learning. In most early attempts, a typical convolutional network is employed to recognize hand gestures from visual frames. R. Sharma et al. trained a machine learning model using 80000 distinct numeric signs with more than 500 photos per sign [6]. Their system technique includes a pre-processed picture training database for a hand-detection system and a gesture recognition system. Before training the machine learning model, image pre-processing comprised feature extraction to standardize the input data. The photos are transformed to grayscale for improved object contour while keeping a consistent resolution and then flattened into fewer one-dimensional components [7]. The feature extraction approach extracts certain pixel data characteristics from pictures and feeds them to CNN for quicker training and more accurate prediction. W. Liu et al. conducted hand tracking in 2D, and 3D space [8]. They obtained a classification accuracy of about 98% by employing skin saliency, which extracts skin tones within a particular range for improved feature extraction.

In addition, the YOLO (You Only Look Once) technique, introduced by Joseph Redmon et al. in 2016 [9], has gained popularity for its rapid and precise performance, making it an ideal choice for real-time applications such as sign language gesture recognition. One study by Y. Song et al. used the YOLO approach to hand gesture recognition in real-time applications [10]. They trained the model using a sizable dataset of hand gesture images, which enabled them to reach a high accuracy rate of 99.4%. This illustrates the power of YOLO and deep learning approaches for precise and efficient hand motion identification. In a similar vein, researchers from the University of Science and Technology Beijing recently published one of the most recent studies on sign language action detection in a paper titled "Sign Language Action Detection based on Multi-Branch Convolutional Neural Network with Spatiotemporal Attention Mechanism [11]. Convolutional neural network (CNN) with a spatiotemporal attention mechanism was developed as a unique approach for sign language action identification. The proposed method produced innovative findings on a benchmark dataset of Chinese Sign Language gestures. The spatiotemporal attention mechanism allowed the network to concentrate on the most instructive portions of the input sequence, while the multi-branch CNN architecture was created to collect both local and global data contained in the sign language video frames. On the same dataset, the proposed method outperformed other cutting-edge algorithms with an accuracy rate of 93.7%.

Hence, ML models require an enormous dataset as well as a difficult technique including significant mathematical processing. Image pre-processing is critical in the gesture tracking process. As a result, for our research, we chose Mediapipe, a Google open-source framework capable of reliably recognizing human body parts. Moreover, in collaboration with Mediapipe and other decencies packages, LSTM, CNN, and SVM machine learning models were chosen for training data and real-time testing.

Chapter 3
System Design

This system explores techniques for training and forecasting the 45 Dynamic American Sign Language (ASL) Gestures on Word-Level to train in various ML algorithms and predict results. A video and image classification approach are investigated to accomplish this task. The sequences of data are used to evaluate and generate a prediction based on its detections using Mediapipe Keypoints with help of OpenCV. The system uses model evaluation to check the model validation, and model training plays a big part in the system as it has different Machine Learning (ML) algorithms like SVM, CNN and LSTM to seek the result for each gesture. This chapter describes steps taken to complete our system to get the best result wanted.

The design can be divided into six main categories:

(i) Dataset Collection, (ii) Holistic Keypoints Extraction, (iii) Labelling and Pre-Processing, (iv) Model Building; (v) Model evaluation; and (vi) Model Training

3.1 Dataset Collection

To accurately recognize gestures within input video feed, it is crucial to have suitable datasets collection for training machine learning models. These datasets enable the model to learn and improve its accuracy over time. For our specific model, we rely on our video data collected through webcam using OpenCV and MediaPipe. The following sections detail the methods used to collect and curate these datasets for use in our SL Action Recognition model.

3.1.1 Dataset Generation

In this project we are using 45 word-level action recognition to text. With the aim of constructing our model, we used video stream data as our input by using Open-CV (Open-Source Computer Vision). It provides a variety of functions to perform common image processing operations Open-Source

The captured video stream is then segmented into frames, with each gesture prediction based on a video sequence length of 30 frames. As each SL gesture will have (30 frames) so in total for 45 SL gesture we have (30*45) 1350 frames loaded as input for the model. Our model can be fine-tuned to operate on both series of input still images and frames from a video stream.

```
cap = cv2.VideoCapture(0) [0 denotes the webcam configuration for
laptop]
no_sequences = 30       # thirty videos worth of data
sequence_length = 30    # videos to be 30 frames in length
```

To interpret the Sign languages (SL), we used Mediapipe Holistic model. MediaPipe Holistic is a machine learning architecture provided by Google's MediaPipe framework, which is designed to perform real-time, multi-modal understanding of human beings. It integrates several computer vision and machine learning models to estimate 3D hand, face, and body landmarks, as well as pose detection. We used the model to apply deep neural networks (such as SVM, CNN and LSTM). It is optimized for real-time performance. The model's ability to track multiple body parts simultaneously and in real-time makes it well-suited for sign language recognition.

```
mp_holistic = mp.solutions.holistic      # holistic pipeline model
mp_drawing = mp.solutions.drawing_utils  # drawing utilities
```

The Mediapipe keypoints are utilized to generate frame values. To classify an SL gesture accurately, 30 frames have been used in our project, which represent 30 distinct sets of keypoints that precede the action. This process does not actually detect the action, as it only examines a single frame. Therefore, to identify a specific action in a video, 30 separate frames or 30*1662 keypoints (section 3.1.2) are required as our input data for each action.

3.1.2 Dataset Preparation using MediaPipe Holistic Keypoints

As we have used MediaPipe Holistic that utilizes the Pose, Face, and Hand Landmark models in MediaPipe Pose (MPP) as shown in Figure 2, MediaPipe Face Mesh and MediaPipe Hands respectively to generate a total of 543 landmarks (468 face landmarks, 33 pose landmarks and 21 landmarks per hand as shown in Figure 1) [12].

- **Face:** 468 landmarks * 3 (x, y, z) = 1404 keypoints
- **Left hand:** 21 landmarks * 3 (x, y, z) = 63 keypoints
- **Right hand:** 21 landmarks * 3 (x, y, z) = 63 keypoints

- **Pose landmarks:** 33 landmarks * 4 (x, y, z, visibility) = 132 keypoints
- Total of 1404 + 63 + 63 + 132 = **1662 keypoints**

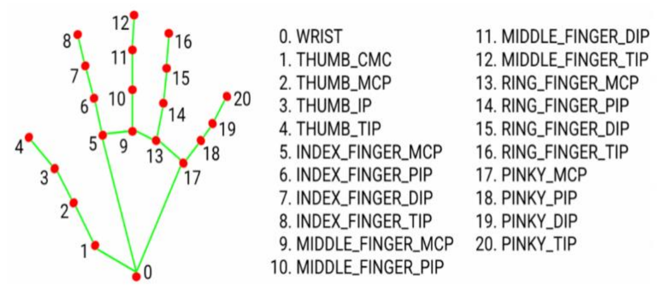


Figure 1: Definition of Landmarks in Mediapipe Hand Landmark model bundle

Figure 1: Definition of landmarks in MediaPipe Hand Landmark model bundle

Med MediaPipe provides estimations of 2D human joint coordinates in each frame of the video. The framework offers pipelines for cognitive data processing. Additionally, when utilizing normalized coordinates for pose estimation, it is necessary to multiply the y-axis pixel values with the inverse ratio [13]. Among the estimated MPP landmarks, we used all 33 landmarks to estimate arbitrary poses and motions, as shown in Figure 2.

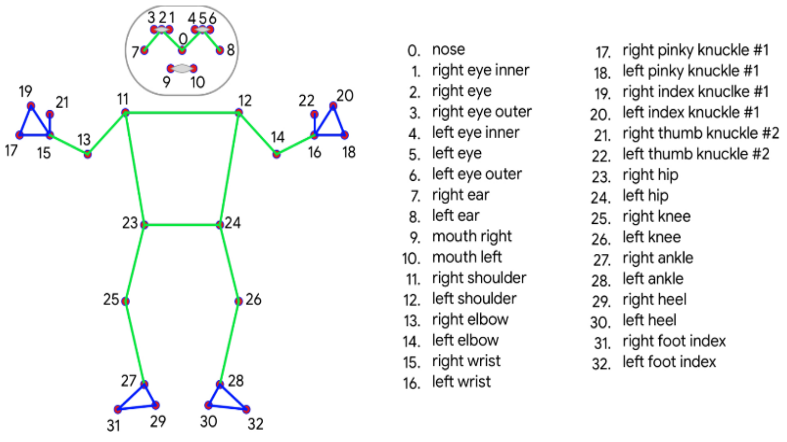


Figure 2: Definition of landmarks in Mediapipe Pose

3.1.3 Dataset Folder Setup

Our project using 45 word-level SL Gestures with each action generation requires 30 sequences of 30 frames each considering our 45 actions as NumPy Array (np.array).

We set our file path where the 45 SL gestures with NumPy arrays are located and define the actions to be detected using machine learning models. It then loops over each action and sequence to read the data from the specified file path and organize it into actions. It attempts to create a directory for each action and sequence within the specified DATA_PATH folder using the **os.makedirs()** function, skipping over any directories that already exist. The graphical representation of the folder setup is shown in Figure 3.

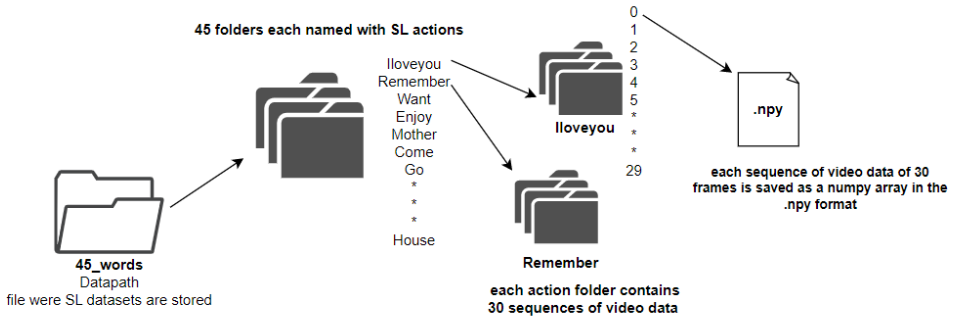

45 folders each named with SL actions
Iloveyou
Remember
Want
Enjoy
Mother
Come
Go
*
*
*
House
Iloveyou
0
1
2
3
4
5
*
*
*
29
.npy
each sequence of video data of 30 frames is saved as a numpy array in the .npy format
45_words
Datapath
file were SL datasets are stored
Remember
each action folder contains 30 sequences of video data

Figure 3: Graphical representation of the folder setup for SLGR

We organized the data into sequenced frames and separated action categories hence the data can be more easily fed into our ML models for training and evaluation.

```
actions = np.array(['Iloveyou', 'Remember', 'Want', 'Father',
'Mother', 'Hungry', 'Cat', 'House', 'Hear', 'Book', 'Water', 'Help',
'Important', 'School', 'Enjoy', 'Why', 'Write', 'Teacher', 'See',
'Quiet', 'You', 'Boy', 'Girl', 'Beautiful', 'Thankyou', 'No', 'Yes',
'Please', 'Babysitter', 'Birthday', 'Come', 'Doctor', 'Eat', 'Give',
'Go', 'Goodbye', 'Hello', 'How', 'Man', 'Name', 'Stop', 'Shirt',
'What', 'World', 'Woman'])
```

During the iteration over each of the 45 actions, our code loops to read the sequences of frames using **cap.read()** and drew landmarks on the frames, using the MediaPipe Holistic model's **draw_landmarks()**. This was done in order to extract and export the landmark **keypoints as a NumPy array.**

Additionally, to ensure accurate and consistent landmark detection, we applied a wait logic **cv2.waitKey(1000)** to collect the landmarks smoothly.

All the steps were easily achieved by **cv2.imshow()** that was used to display our video in a window. The procedure for collection is depicted in Figure 4.

```
for i, sequence in enumerate(sequences):
    np.save(f'sequence_{i}.npy', sequence)
```

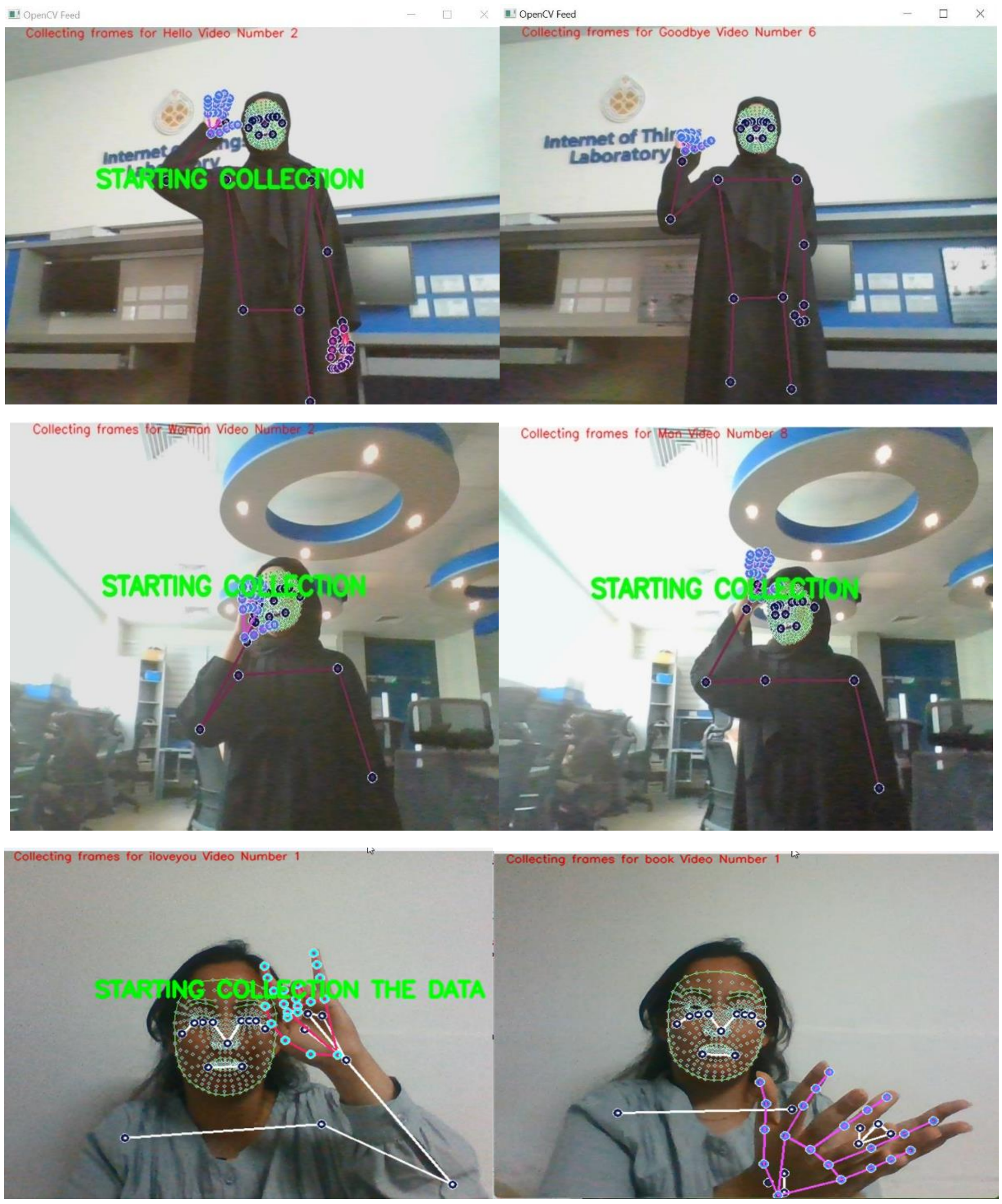

OpenCV Feed
Collecting frames for Hello Video Number 2
STARTING COLLECTION
OpenCV Feed
Collecting frames for Goodbye Video Number 6
STARTING COLLECTION
Collecting frames for Woman Video Number 2
STARTING COLLECTION
Collecting frames for Man Video Number 8
STARTING COLLECTION
Collecting frames for iloveyou Video Number 1
STARTING COLLECTION THE DATA
Collecting frames for book Video Number 1

Figure 4: Procedure for SLGR collection (Hello, Goodbye, Woman, Man, Iloveyou, Book)

3.2 Holistic Keypoints Extraction

MediaPipe Holistic incorporates pose prediction, which predicts the position of body parts in each frame and serves as an additional (Region of Interest) ROI. This also ensures that the model maintains semantic consistency across the body and its parts by avoiding confusion between the left and right hands or body parts of different individuals in the frame. MediaPipe Holistic takes an RGB (Red, Green, Blue) image and produces landmarks [14] for pose, left and right hand, and face mesh on the most prominent individual recognized in the image as shown in Figure 5.

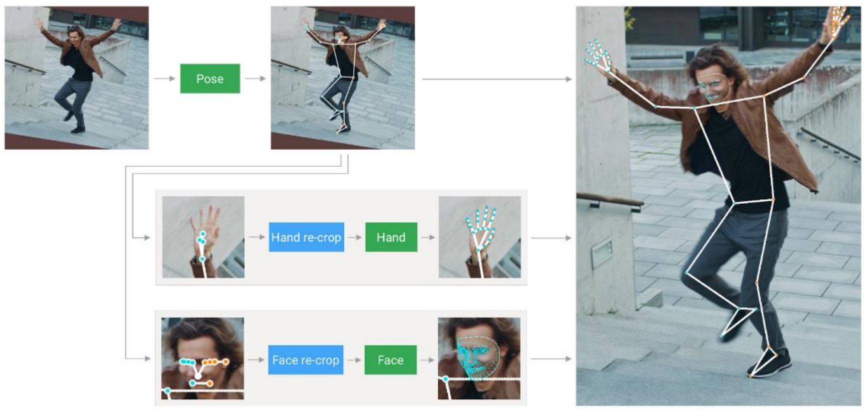


Figure 5: MediaPipe Holistic Pipeline Overview

We extracted our MediaPipe Holistic Keypoints for pose, face, left and right hands using the following code:

```
# total of 1662 features are extracted from each frames
def extract_keypoints(results):
    pose = np.array([[res.x, res.y, res.z, res.visibility] for res
in results.pose_landmarks.landmark]).flatten() if
results.pose_landmarks else np.zeros(33*4)
    face = np.array([[res.x, res.y, res.z] for res in
results.face_landmarks.landmark]).flatten() if
results.face_landmarks else np.zeros(468*3)
    lh = np.array([[res.x, res.y, res.z] for res in
results.left_hand_landmarks.landmark]).flatten() if
results.left_hand_landmarks else np.zeros(21*3)
    rh = np.array([[res.x, res.y, res.z] for res in
results.right_hand_landmarks.landmark]).flatten() if
results.right_hand_landmarks else np.zeros(21*3)
    return np.concatenate([pose, face, lh, rh])
```

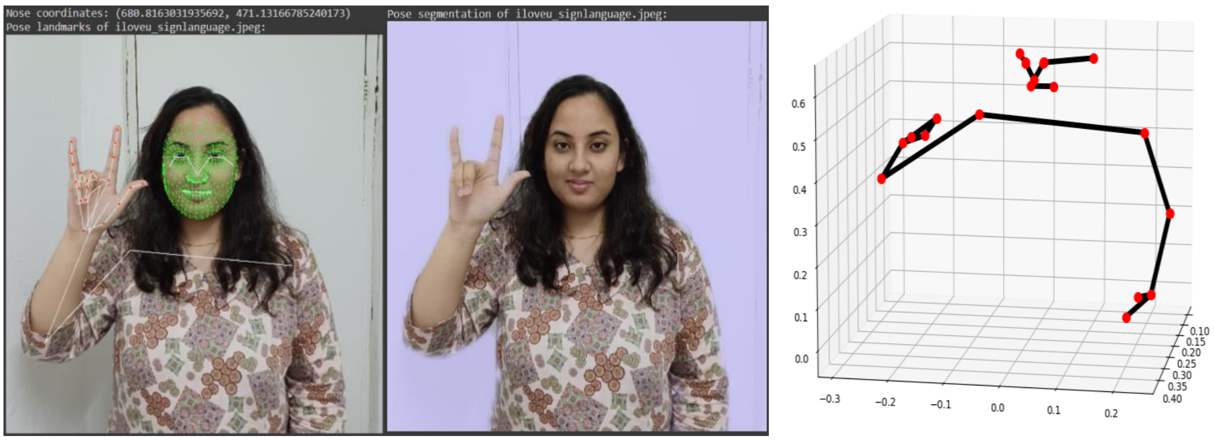
For instance, we can see on **pose** code it extracts the 3D coordinates and visibility of each pose landmark from the results.pose_landmarks object, which is generated by the Mediapipe Holistic model finally converted into a NumPy array. The array allows for easy processing and manipulation of the landmark data for further analysis.

The pose_landmarks object contains information about the pose of the body, including the position and orientation of the head, torso, and limbs. It uses a list comprehension to iterate over each Landmark object in the pose_landmarks object and extract the X, Y, and Z coordinates, as well as the visibility of each landmark.
The resulting array is flattened using the flatten() method to create a one-dimensional array that can be easily processed by other functions or models. If the pose_landmarks object is not

present in results, then the array is initialized as an array of zeros with a length of 33*4, which corresponds to the number of pose landmarks multiplied by the number of features (X, Y, Z, and visibility) for each landmark.

3.2.1 Holistic Model Configuration Options

To demonstrate the MediaPipe Holistic Keypoints (section 3.1.2) we used the default Holistic model setting (code below) to draw nose coordinates and the pose landmarks and extract its values to plot a 3D Pose Landmarks for better Visualization. We can also use change setting to `**enable_segmentation=True**` to get pose segmentation on image as shown in Figure 6.



```
Nose coordinates: (680.8163031935692, 471.13166785240173)
Pose landmarks of iloveu_signlanguage.jpeg:
```

```
Pose segmentation of iloveu_signlanguage.jpeg:
```

Figure 6: Holistic Model features using Draw, Segmentation Landmarks and 3D Pose

Default Settings

```
Holistic (self, static_image_mode=False, model_complexity=1,
smooth_landmarks=True, enable_segmentation=False,
smooth_segmentation=True, refine_face_landmarks=False,
min_detection_confidence=0.5, min_tracking_confidence=0.5)
```

In our SL recognition project, we kept everything as default other than the following:

```
STATIC_IMAGE_MODE = True
ENABLE_SEGMENTATION = True
```

STATIC_IMAGE_MODE is set to true; person detection runs every input image. ENABLE_SEGMENTATION is set to true; mask has the same width and height as the input image, and contains values in [0.0, 1.0] where 1.0 and 0.0 indicate high certainty of a "human" and "background" pixel, respectively.

```
min_detection_confidence = 0.7
min_tracking_confidence = 0.7
```

Minimum confidence value ([0.0, 1.0]) from the person-detection model for the detection to be considered successful. Minimum confidence value ([0.0, 1.0]) from the landmark-tracking model for the pose landmarks to be considered tracked successfully, or otherwise person detection will be invoked automatically on the next input image [15]. Setting it to a higher value can increase robustness of the solution, at the expense of a higher latency (default=0.5).

3.3 Labelling and Pre-processing

During the stage of KeyPoint extraction from frames using OpenCV functions our collected frames are in the default color format of BGR (blue, green, red), which is different from the more commonly used RGB (red, green, blue) color format. This is because OpenCV was originally developed for computer vision applications, where the BGR format is more commonly used. Hence, we converted our images using OpenCV provides tools from BGR images to RGB.

```
#Color Conversion using Opencv
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
```

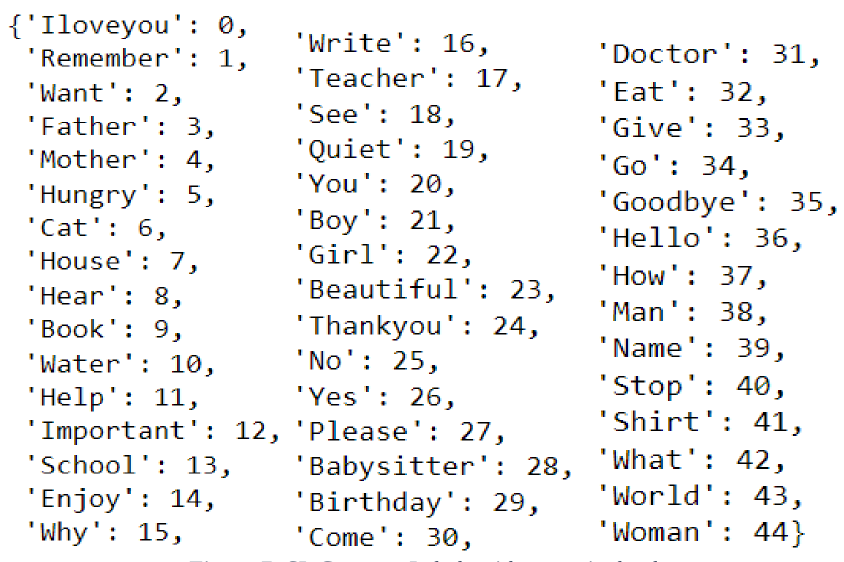
3.3.1 Action Labelling

Labels are used in our action recognition tasks to indicate the specific action being performed in each video or sequence of frames. Assigning a label to each action allows us to train machine learning models to classify actions based on their visual representation.

We created a dictionary called **label_map** that assigns a unique number to each label in the actions list (see section 3.1.3). The numbers are assigned in ascending order starting from 0.

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

By using this dictionary, we can easily convert the label strings to their corresponding numerical values shown on Figure 7. For instance, if we have a label string 'Goodbye', we can use the **label_map** to get its numerical value which is 35 in our project.



```
{'Iloveyou': 0,
 'Remember': 1,
 'Want': 2,
 'Father': 3,
 'Mother': 4,
 'Hungry': 5,
 'Cat': 6,
 'House': 7,
 'Hear': 8,
 'Book': 9,
 'Water': 10,
 'Help': 11,
 'Important': 12,
 'School': 13,
 'Enjoy': 14,
 'Why': 15,
 'Write': 16,
 'Teacher': 17,
 'See': 18,
 'Quiet': 19,
 'You': 20,
 'Boy': 21,
 'Girl': 22,
 'Beautiful': 23,
 'Thankyou': 24,
 'No': 25,
 'Yes': 26,
 'Please': 27,
 'Babysitter': 28,
 'Birthday': 29,
 'Come': 30,
 'Doctor': 31,
 'Eat': 32,
 'Give': 33,
 'Go': 34,
 'Goodbye': 35,
 'Hello': 36,
 'How': 37,
 'Man': 38,
 'Name': 39,
 'Stop': 40,
 'Shirt': 41,
 'What': 42,
 'World': 43,
 'Woman': 44}
```

Figure 7: SL Gestures Labels with numerical values

By using labels, we can evaluate the performance of our machine learning model by comparing its predicted labels with the ground truth labels. This allows us to measure the accuracy of our model and helps improve.

3.3.2 Sequence Loading in Data path

After reading a set of video sequences from **"DATA_PATH"** directory and we created a list of NumPy arrays, where each NumPy array represents a sequence of frames from a video. The actions list (see on Figure 7) contains the names of the subdirectories in the directory, with each subdirectory containing the frames for a particular action. Each sequence of frames is represented by a directory name, which is converted to an integer using the **astype(int)** method (see Figure 3 on section 3.1.3).

```
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])
```

The NumPy arrays for each frame are then added to a list called **window**, which represents a single sequence of frames. After all the frames in a sequence have been added to the **window** list, the window list is added to a sequences list, which contains all the sequences for all the actions.

```
# converting the sequences and labels to numpy arrays for training
X = np.array(sequences)
Y = np.array(labels).astype(int)
```

Here, X finally represents the total images (30*45 = 1350), and Y represents the 45 Action Labels. The shape of the frame sequences is (1350, 30, 1662), 30 is for number of frames, 1662 is total keypoints (see section 3.1.2).

3.3.3 Splitting Data for Training Models

We used **train_test_split** function from the model_selection module of the scikit-learn library for splitting our SL datasets into training and testing subsets. It randomly dividesd our dataset into two parts: one for training the model and another for evaluating or testing its performance.

```
# Split the data into Train ( 70% ) and Test (20%)
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X , Y,
test_size=0.2, random_state = 42)
```

Test size is set as **0.2**(the proportion of the dataset to be used for testing) which divided 70% SL datasets for training and 20% for Testing and set the random seed to 42 (to ensure reproducibility of the split). The resulting X_train, X_test, Y_train, and Y_test data will be used to train and evaluate.

```
Number of training images:  1080
Number of testing images:  270
X_train.shape = (1080, 30, 1662), X_test.shape = (270, 30, 1662)
Y_train.shape = (1080), Y_test.shape = (270)
```

3.3.4 Data Standardization

Data standardization is scaling and adjusting the data to attain a mean and standard deviation of 0 and 1, respectively. Scaling and normalizing data before training is a key step in our project. It is necessary when the features (input variables) have different units or scales, these can be skipped in case of clean datasets. It is useful when the distribution of the data is skewed (refers to data that is not symmetrical).
The goal to perform Standardization is to bring down all the features to a common scale without distorting the differences in the range of values.
Fit_transform() and **trasnform()** both are methods of class **sklearn.preprocessing.StandardScaler()** it is used in training and testing dataset, respectively.

```
X_train[:, :, i] = scalers[i].fit_transform(X_train[:, :, i])
#training dataset
```

```
X_test[:, :, i] = scalers[i].transform(X_test[:, : , i])
#testing dataset
```

fit_transform learns the scaling parameters from training data. Model built will learn the Mean and Variance of the features of the training set [16]. These learned parameters are later used to scale our testing datasets.
As we do not want to be biased towards a particular feature and want our test data to be completely new and a surprise set to our model, we used transform(). Eventually, after all this Labelling and pre-processing of datasets with our SL datasets can finally be used to train on our various ML models (SVM, CNN, LSTM).

3.4 Model Building

Model building is the process of machine learning in which the prepared dataset is provided to a machine-learning algorithm to perform learning from the training data. The training data contains a set of features and a target variable, which contain an accurate value in the case of regression and a categorical value in classification. The process of building a ML model using our processed SL gesture dataset involves the following steps: (Figure 8)

- Input video (Uses OpenCV's VideoCapture to load the input video)
- Extract frames from the video
- Extract the Keypoints and Landmarks using Mediapipe (Pose_Landamarks, Righthand_Landmarks, Lefthand_Landmarks,)
- Label the frames (Object detection and Segmentation)
- Apply classifying algorithms to the labeled frames (Using machine learning models such as CNN (Convolutional Neural Networks), SVM (Support Vector Machine) and LSTM (Long Short-Term Memory)
- Predict the class of each frame (presence or absence of certain gestures)
- Aggregate the frame-level predictions to make a prediction for the entire video (By taking the majority vote or averaging the predictions)

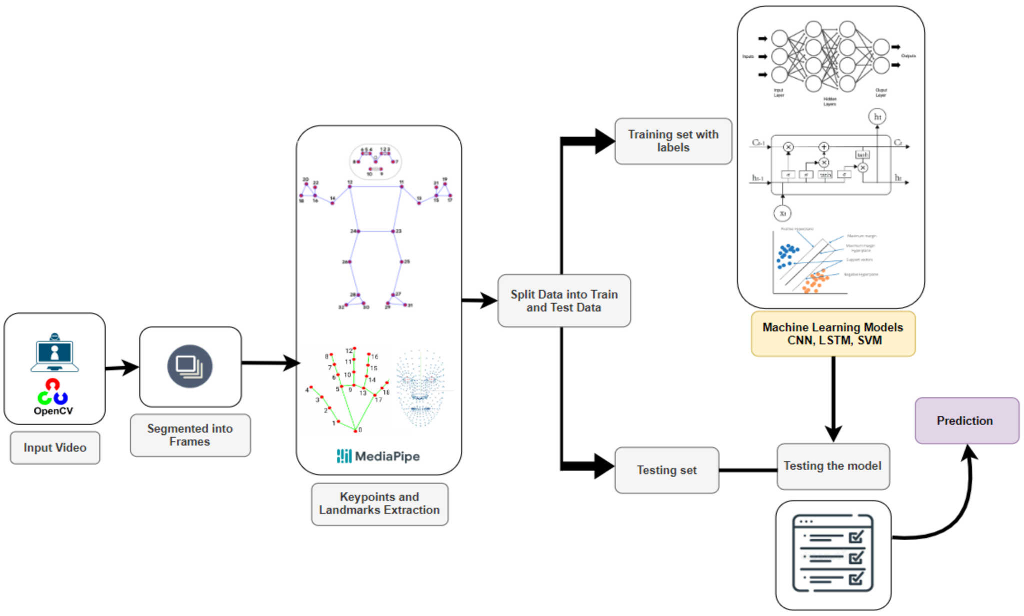


Figure 8: An Overview of modeling and prediction of our SLGR Model

After building the ML model with different ML Algorithms and neural networks their weights are saved. Saving machine learning models is a crucial step in the machine learning workflow because it allows us to reuse the trained models for future predictions or other purposes, without having to retrain the model from scratch.

3.4.1 ML Model Algorithms

There are several machine learning algorithms that can be used to build predictive models. We used the SVM, CNN and LSTM Models to predict our SL Gestures. During the Model building we closely notice the hyperparameters. These are parameters set by the user before training a machine learning model, rather than learned from the training data itself, control the behavior of the model and affect its performance and behavior.

a) **Support Vector Machines (SVMs)** is a supervised learning algorithm for classification and regression. It finds the best hyperplane to separate data into classes with the largest margin. It is robust against noise and can handle non-linear data using the kernel trick. SVM classifier is used in image and text classification but can be sensitive to hyperparameters and slow for large datasets.

```
SVM_classifier.fit(X_train, Y_train)
```

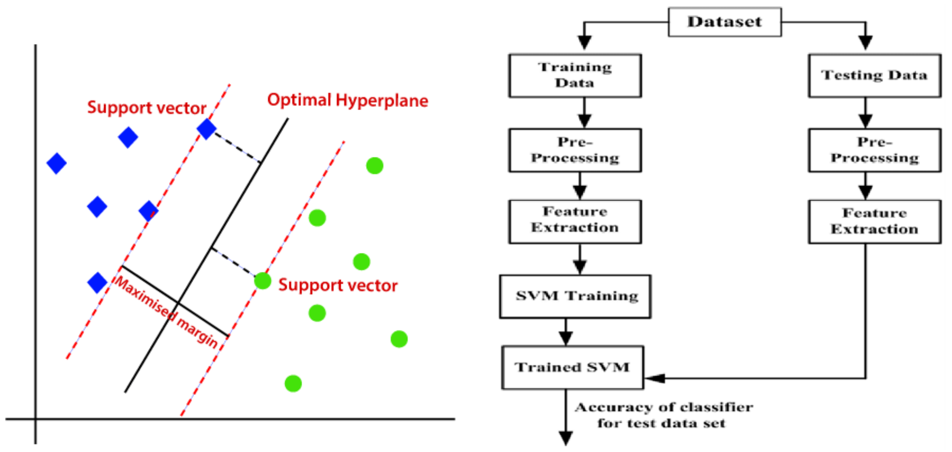


Figure 9: An Overview of ML SVM Classifier

The SVC class from the SVM module is used to create an instance of an SVM classifier. The classifier is then trained on the training data using the fit method of the SVC class.

b) **Convolutional Neural Networks (CNN)** is used for image and Video processing and are particularly effective in tasks like image classification, object detection and face recognition. CNN are deep learning algorithms used for image and video processing tasks.

We used the 3 Layer CNN model (Figure 10) with filters to extract features from input images, down sampled by pooling layers, and use fully connected Dense layers to perform classification. They achieve state-of-the-art performance in computer vision tasks. Our input shapes are (Video time frames: 30 as n_timesteps and Keypoint Features: 1662 as n_features).

```
# model Summary
    model = Sequential()
    model.add(Conv2D(32, (5, 5), activation='relu',
input_shape=(n_timesteps,n_features, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.summary()
```

We used **ReLU** as our activation function as it is commonly used in neural networks and deep learning models, which sets all negative values to zero and passes all positive values through unchanged. Addition to **softmax** activation function commonly used in neural networks for multi-class classification problems, which takes a vector of real numbers as input and returns a probability distribution over the vector elements. The Activation Function assists in limiting the output value of a neuron within a specific range, as needed. If the limit is not set, then the output will reach remarkably high magnitudes. Most of the models based on real-life concepts are non-linear so the activation functions. ReLu and softmax activation function act like hyperparameters (see section 3.5).

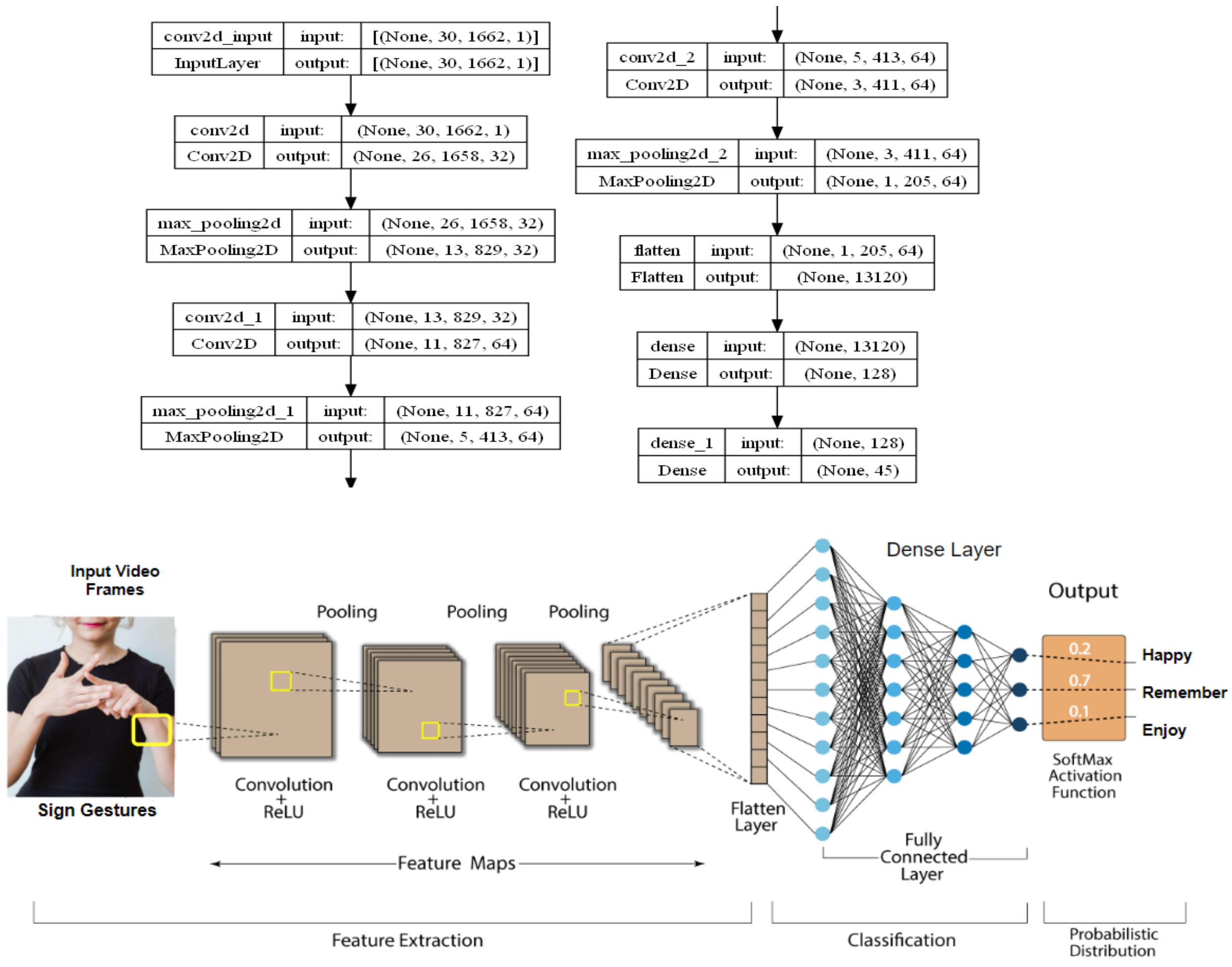

conv2d_input | InputLayer | input: [(None, 30, 1662, 1)] | output: [(None, 30, 1662, 1)]
conv2d | Conv2D | input: (None, 30, 1662, 1) | output: (None, 26, 1658, 32)
max_pooling2d | MaxPooling2D | input: (None, 26, 1658, 32) | output: (None, 13, 829, 32)
conv2d_1 | Conv2D | input: (None, 13, 829, 32) | output: (None, 11, 827, 64)
max_pooling2d_1 | MaxPooling2D | input: (None, 11, 827, 64) | output: (None, 5, 413, 64)
conv2d_2 | Conv2D | input: (None, 5, 413, 64) | output: (None, 3, 411, 64)
max_pooling2d_2 | MaxPooling2D | input: (None, 3, 411, 64) | output: (None, 1, 205, 64)
flatten | Flatten | input: (None, 1, 205, 64) | output: (None, 13120)
dense | Dense | input: (None, 13120) | output: (None, 128)
dense_1 | Dense | input: (None, 128) | output: (None, 45)
Input Video Frames
Sign Gestures
Pooling
Pooling
Pooling
Convolution + ReLU
Convolution + ReLU
Convolution + ReLU
Feature Maps
Flatten Layer
Dense Layer
Fully Connected Layer
Output
0.2
0.7
0.1
Happy
Remember
Enjoy
SoftMax Activation Function
Feature Extraction
Classification
Probabilistic Distribution

Figure 10: 3 Layers CNN Model summary and structure

```
# Specify loss function, optimizer, and metrics values to the CNN
CNN_model.compile(loss = 'categorical_crossentropy', optimizer =
'Adam', metrics = ["accuracy"])
 history = CNN_model.fit(train_X, train_Y, epochs=epochs,
batch_size=batch_size, verbose=verbose, validation_split=0.1,
shuffle = True)
```

The compile function is applied using Keras API, to specify the loss function, optimizer, and evaluation metric(s) to be used during training. In our case, the categorical cross-entropy loss function, Adam optimizer, and accuracy metric are specified. Fit function is then used to train the CNN model on the training dataset (train_X and train_Y) for a specified number of epochs and batch size which are some of the hyperparameter's in our CNN model (see section 3.5).

c) **Long Short-Term Memory (LSTM)** is used for sequence modelling tasks such as language translation, speech and image recognition and sentiment analysis. LSTM networks are widely used in natural language processing, speech recognition, video analysis, and other applications where long-term dependencies and sequential data are present. They have been shown to be effective in modeling complex temporal patterns and outperform traditional RNNs (Recurrent Neural Networks) in many tasks.

- LSTM layer with 120 units, the input shape of LSTM model is same as the CNN to maintain consistency.
- Dropout layer with a rate of 0.5, which randomly drops 50% of the input units during training to prevent overfitting.
- Dense layer with 100 units and ReLU activation function.
- Dense layer with n_outputs units (Action words: 45) and softmax activation function.

```
model = Sequential()
model.add(LSTM(120, input_shape=(n_timesteps,n_features)))
model.add(Dropout(0.5))
model.add(Dense(100, activation='relu'))
model.add(Dense(n_outputs, activation='softmax'))
model.summary()
return model
```

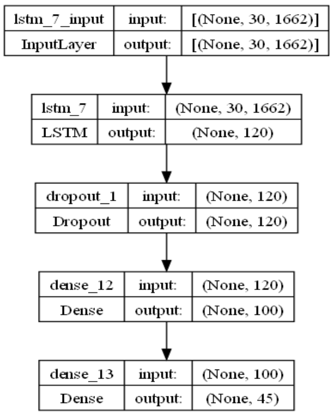
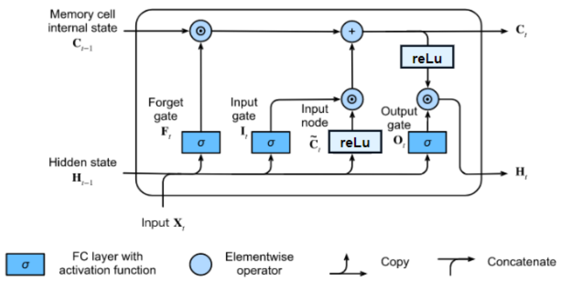

lstm_7_input | InputLayer | input: | [(None, 30, 1662)] | output: | [(None, 30, 1662)]
lstm_7 | LSTM | input: | (None, 30, 1662) | output: | (None, 120)
dropout_1 | Dropout | input: | (None, 120) | output: | (None, 120)
dense_12 | Dense | input: | (None, 120) | output: | (None, 100)
dense_13 | Dense | input: | (None, 100) | output: | (None, 45)


Figure 11: LSTM model summary and constructed structure

The compile function specified loss function, optimizer, and evaluation metric(s) is same as CNN used during training. The fit function is then used to train the LSTM model on the training dataset (train_X and train_Y) for a specified number of epochs and batch size which are some of the hyperparameter's (see section 3.5).

3.5 Model Evaluation

Model evaluation is a crucial step in the machine learning workflow, as it allows us to assess the performance of our trained model and determine whether it is suitable for the task at hand. These are the metrics we used for evaluating our machine learning models:

- **epochs** parameter specifies the number of times to iterate over the entire training dataset
- **batch_size** specifies the number of samples to use in each iteration.
- verbose parameter controls the amount of logging output during training, with a value of 1 indicating that progress updates should be printed to the console.
- **validation_split** parameter specifies the fraction of the training dataset to use for validation during training, while the shuffle parameter controls whether the training data should be shuffled before each epoch.
- **Activation functions** used are reLu and softmax (see section 3.4.1).

There are various approaches to performing a model evaluation, and each one of them has its advantages. We have used hold-out validation with 80% of the data used for model training, and the rest of the 20% data is used for model validation

3.6 Model Training

To perform model training, we have designed a few sets of experiments. The first experiment was to check how the number of frames and the number word-level SL can determine changes in our various ML model algorithms only using features obtained from "MediaPipe" can provide variedness in our predictions.

3.6.1 Initial Training

During the initial experimental training stages, we focused on using our Sign Language (SL) datasets with our Long Short-Term Memory (LSTM) model. We used a smaller number of SL gestures and video frames to assess how the model was responding to the available information. This allowed us to analyze the model's behavior and performance with a limited dataset before scaling up to more extensive datasets. By evaluating the model with a smaller dataset, we could identify any potential issues early and adjust the model accordingly. This approach helped us to optimize the model's performance before applying it to larger datasets and more complex tasks (see Figure 12).

The goal is to have both high training accuracy and high validation accuracy. High training accuracy indicates that the model has learned to fit the training data well, while high validation accuracy indicates that the model can generalize well to new data. In Figure 12, the training accuracy is high, but the validation accuracy is low, it indicates that the model is overfitting to the training data and is not able to generalize well. In this case, the model needed to be adjusted to reduce overfitting and improve validation accuracy.

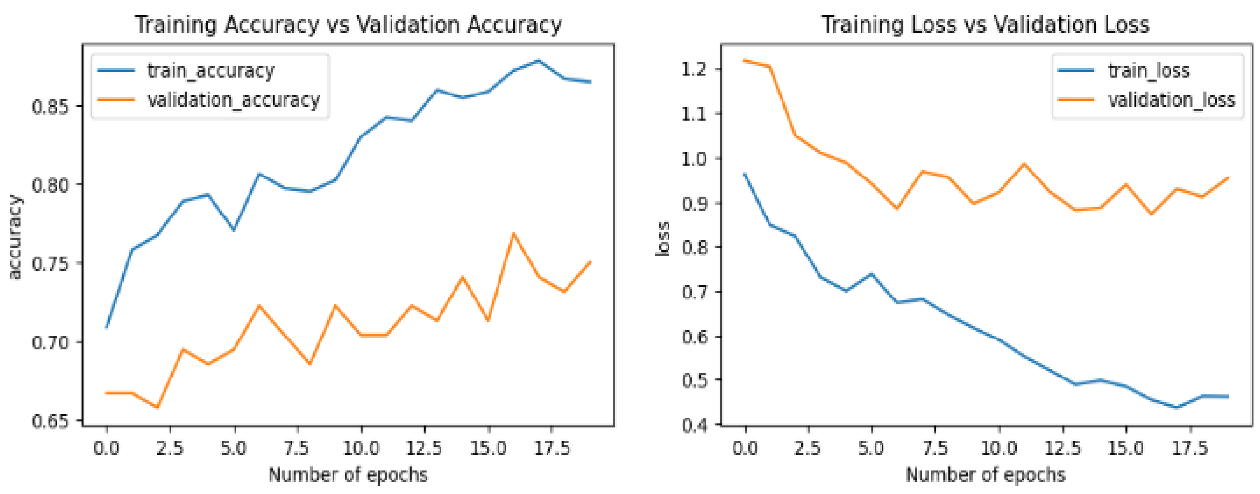


Figure 12: Training of LSTM Overfitting Model Summary with 16 words and 30 frames

3.6.2 CNN and LSTM Model Training

Training a ML model is a complex process that requires expertise in deep learning, data preprocessing, and optimization algorithms. Careful consideration of the model architecture, hyperparameters, and data preparation is critical to achieving a high-performing model.

For our project's CNN model training, we set the hyperparameters to epoch=15 and batch_size=32 (described in section 3.5). We can notice from Figure 13 that our CNN model has similar training and validation accuracy is a good sign that the model is performing well and is not overfitting to the training data. However, it is important to continue monitoring the model's performance and adjusting its architecture and hyperparameters as needed to optimize its performance. It also indicates that the model architecture and hyperparameters are well-suited to the problem being addressed.

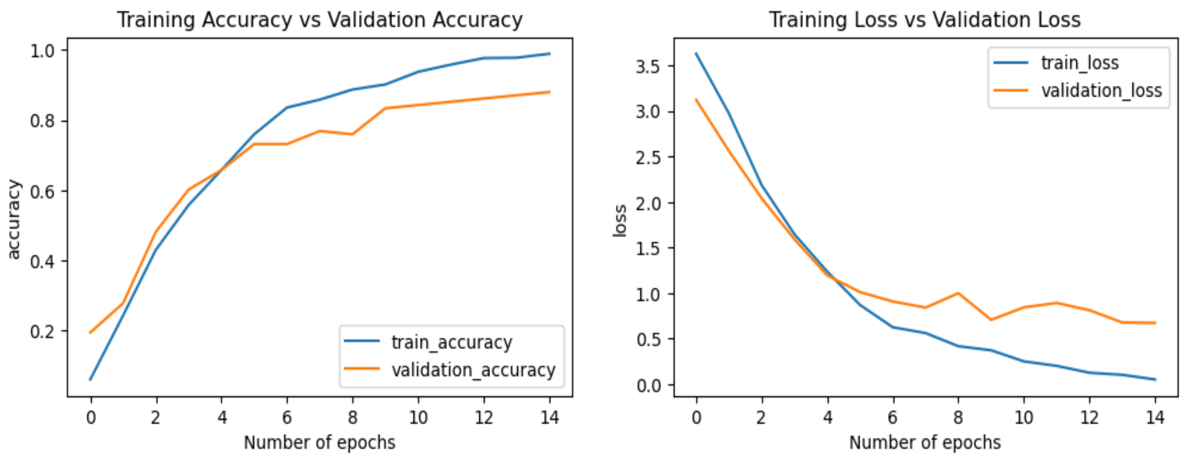


Figure 13: Learning and Loss Curve of CNN Model

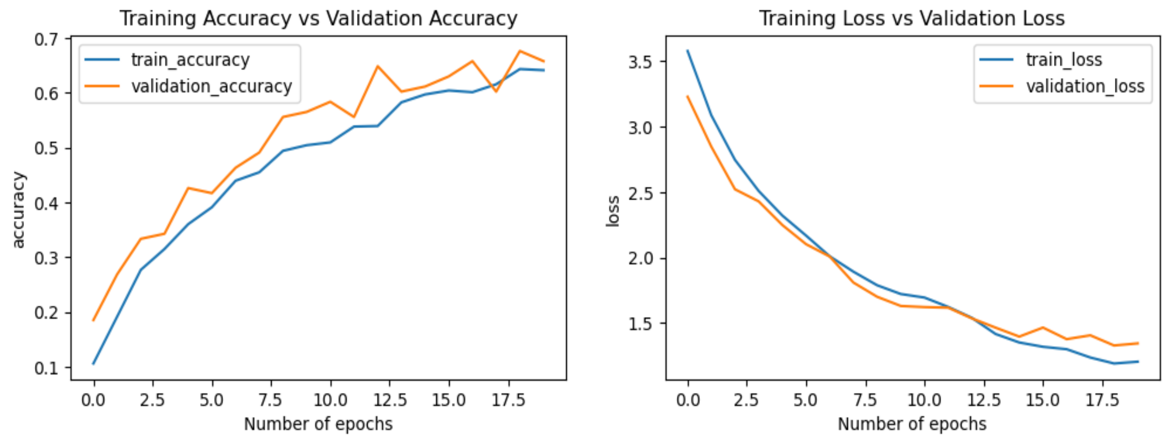


Figure 14: Learning and Loss Curve of LSTM Model

Similarly, in Figure 14 our project's LSTM model training, we set the hyperparameters to epoch=20 and batch_size=32. The learning and Loss Curve is a visual representation of how the accuracy of the model changes as the training progresses. Learning Curve typically shows the training accuracy and validation accuracy on the y-axis and the number of training epochs or iterations on the x-axis. T

Loss curve is a useful tool for evaluating the performance of an ML model because it measures how well it can minimize its loss on the training and validation sets.

Chapter 4
System Implementation

This chapter focuses on the development of a predictive machine learning (ML) model for Sign Language Gesture Recognition (SLGR), using Support Vector Machines (SVM), Convolutional Neural Networks (CNN), and Long Short-Term Memory (LSTM). The project aims to address the challenge of predicting SL gestures accurately, particularly in larger datasets or with more volatility. Despite the difficulty in anticipating SL gestures, machine learning models attempt to predict with some accuracy. This chapter discusses the tools and technologies used in the development of the predictive ML model for SLGR.

4.1 Software Implementation

4.1.1 Tools and Libraries

A machine learning library is a collection of tools and functions that allow developers and data scientists to build and train machine learning models see Figure 8. These libraries provide a high-level interface for working with machine learning algorithms and models, making it easier to experiment with different approaches and techniques.

Our project is developed in Python 3.9. Python is one of the most powerful programming languages, can easily achieve a wide range of complicated programming goals due to its extensive feature set and adaptability.

- OpenCV (Open-Source Computer Vision Library) is an open-source software library for computer vision and image processing. This library has more than 2500 algorithms dedicated to computer vision and ML. OpenCV- was used to be a standard foundation for computer vision in our project to speed up the incorporation of machine perception into commercial goods. OpenCV version 4.7.0.72 was used in our project.
- MediaPipe is a cross-platform library developed by Google that delivers fantastic ready-to-use machine-learning solutions for computer vision tasks to construct unique machine-learning solutions for live and streaming video. The Framework and Solutions are part of the MediaPipe Toolkit. MediaPipe was used to set the model by drawing the landmarks and extracting the key points out of it. (0.8.11) MediaPipe version 0.8.11 was used in our project.

- TensorFlow allows easy distribution of work into multiple CPU cores or GPU cores and can even distribute the work to multiple GPUs. TensorFlow version 2.4.1 was used in our project.
- NumPy (Numerical Python) is a Python library for manipulating arrays. It also has functions for working with linear algebra, the Fourier transform, and matrices. NumPy strives to produce array objects that are up to 50 times quicker than typical Python lists. NumPy version 1.19.5 was used in our project.
- The OS module in Python is a component of the computer language's standard library. We imported it to let it create a path for exported data, NumPy arrays, and so on so that we can recover the data on that path to generate labels and train them. The OS module version 0.1.1 was used in our project.
- Matplotlib is a Python package that may be used to generate 2D graphs and plots using Python programs, graphical representation to understand the data before moving it to data-processing and training it for machine learning purposes. It features a module called pyplot that simplifies. It is used with NumPy to give an environment that is a viable open-source alternative to MATLAB. Matplotlib version 3.5.3 was used in our project.
- The Python time module offers several ways to represent time in code, including objects, integers, and texts. Other than displaying time, it also provides functions such as waiting during code execution and assessing code efficiency.
- Scikit-learn (sklearn) is a Python-based machine-learning library that helps in statistical modeling, such as classification, regression, clustering, and dimensionality reduction, via a Python interface. In this project, it was used to divide arrays or matrices into random trains and test subsets. Scikit-learn 0.0.post1 was used in our project.
- Keras is the TensorFlow platform's high-level API. It is created to understand deep learning approaches, such as generating layers for neural networks, provides fundamental abstractions and building blocks for rapidly designing and releasing machine learning applications. We used Keras for our ML model implementation. It provides a python interface of TensorFlow Library especially focused on AI neural networks. Keras version 2.10.0 was used in our project.

4.1.2 Platforms

- Jupyter Notebook is an interactive computing environment that runs on the web. Jupyter notebooks are documents that include software code, computational output, explanatory prose, and rich content. Notebooks enable in-browser code modification and execution and the display of calculation results. An .ipynb extension is used to store a notebook. The Jupyter Notebook project supports a wide range of computer languages, including Julia (Ju), Python (Py), and R.

4.2 Raspberry Pi Implementation

4.2.1 Description and components

The Raspberry Pi 3 Model B is the earliest model of the third-generation Raspberry Pi. It replaced the Raspberry Pi 2 Model B in February 2016 [17]. The Raspberry Pi was selected for this research project as it is an appealing platform for creating and deploying sign language action detection systems due to its affordability, accessibility, portability, flexibility, customizability, and community support. The following equipment were used in order to configure the Raspberry Pi:

1. Raspberry Pi 3 Model B
2. Micro SD card (32 GB) with Raspberry Pi OS installed
3. Micro USB power supply (0.35 A)
4. VNC Viewer (Remote Access)
5. LAN wire
6. The Raspberry Pi Camera Rev 1.3
7. Laptop.

4.2.2 Deployment

Implementing our SLGR system on a Raspberry Pi 3 is a multi-step procedure that demands careful consideration at each stage. To achieve the goal of effectively recognizing and classifying human gestures in sign language, several major steps were taken. Each of these steps that followed in this project for implementation needed on the Raspberry Pi 3 is outlined below:

a) Raspbian Mirror:

A Raspberry Pi mirror, or Raspbian mirror, is a copy of the Raspbian operating system and associated packages hosted on a remote server. Mirrors are used to distribute software and updates more efficiently by allowing users to download files from a server locally.

We operate actions that are needed to run the update command, there were some errors like it does not fetch some of the archive files and to solve this problem, configuring the Raspbian mirror was the best solution for it. Changing it also improves the speed, reliability, and availability of your downloads, while also providing greater flexibility and customization options. The nearest geographical mirror to Bahrain was Iran, depending on the latest official list [19].

- Official Raspbian mirror:

```
deb http://raspbian.raspberrypi.org/raspbian/ bullseye main contrib
non-free rpi
# Uncomment line below then 'apt-get update' to enable 'apt-get
source'
#deb-src http://raspbian.raspberrypi.org/raspbian/ bullseye main
contrib non-free rpi
```

- Modified Raspbian mirror:

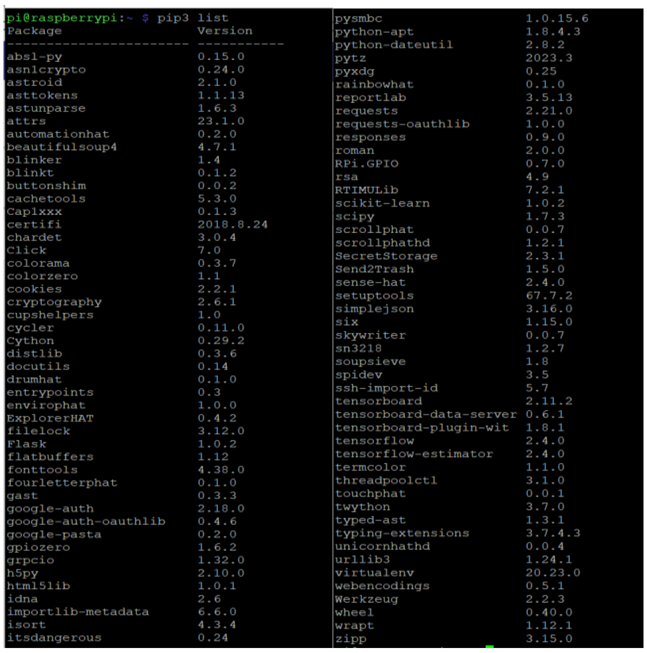
```
dep http://repo.iut.ac.ir/repo/raspbian/raspbian/ bullseye main
contrib non-free rpi
# Uncomment line below then 'apt-get update' to enable 'apt-get
source'
#deb-src http://raspbian.raspberrypi.org/raspbian/ bullseye main
contrib non-free rpi
```

b) PiCamera: Raspberry Pi Camera Board v1.3 is a 5-megapixel camera module with 1080p resolution and infrared capabilities, designed to be used with the Raspberry Pi 3 Model B or other single-board computers. It attaches to the Raspberry Pi via a small socket on the board's upper surface and utilizes the CSI interface for camera interfacing [20]. To use it, we installed the IP camera package and activate Camera Support in raspi-config.

c) Packages and Dependences: Installing additional packages is essential for any machine learning-based project. It is also a good practice to manage project dependencies in a single virtual environment for easier management. However, it's crucial to understand the requirements for each package before installation, as they may depend on the Python and uname (command-line utility in Unix-based OS) of the Raspberry Pi 3 Model B. In this project, the Raspberry Pi 3 default Python version is Python 3.9.2, while the Uname version is armv7l.

Following the downgrade of the Python version, it is necessary to verify that other packages are compatible with the current version.

Figure 15 provides with list of required packages that have been installed, along with the necessary steps for installation:



```
pi@raspberrypi:~ $ pip3 list
Package                  Version
------------------------ ----------
absl-py                  0.15.0
asn1crypto               0.24.0
astroid                  2.1.0
asttokens                1.1.13
astunparse               1.6.3
attrs                    23.1.0
automationhat            0.2.0
beautifulsoup4           4.7.1
blinker                  1.4
blinkt                   0.1.2
buttonshim               0.0.2
cachetools               5.3.0
Cap1xxx                  0.1.3
certifi                  2018.8.24
chardet                  3.0.4
Click                    7.0
colorama                 0.3.7
colorzero                1.1
cookies                  2.2.1
cryptography             2.6.1
cupshelpers              1.0
cycler                   0.11.0
Cython                   0.29.2
distlib                  0.3.6
docutils                 0.14
drumhat                  0.1.0
entrypoints              0.3
envirophat               1.0.0
ExplorerHAT              0.4.2
filelock                 3.12.0
Flask                    1.0.2
flatbuffers              1.12
fonttools                4.38.0
fourletterphat           0.1.0
gast                     0.3.3
google-auth              2.18.0
google-auth-oauthlib     0.4.6
google-pasta             0.2.0
gpiozero                 1.6.2
grpcio                   1.32.0
h5py                     2.10.0
html5lib                 1.0.1
idna                     2.6
importlib-metadata       6.6.0
isort                    4.3.4
itsdangerous             0.24
pysmbc                   1.0.15.6
python-apt               1.8.4.3
python-dateutil          2.8.2
pytz                     2023.3
pyxdg                    0.25
rainbowhat               0.1.0
reportlab                3.5.13
requests                 2.21.0
requests-oauthlib        1.0.0
responses                0.9.0
roman                    2.0.0
RPi.GPIO                 0.7.0
rsa                      4.9
RTIMULib                 7.2.1
scikit-learn             1.0.2
scipy                    1.7.3
scrollphat               0.0.7
scrollphathd             1.2.1
SecretStorage            2.3.1
Send2Trash               1.5.0
sense-hat                2.4.0
setuptools               67.7.2
simplejson               3.16.0
six                      1.15.0
skywriter                0.0.7
sn3218                   1.2.7
soupsieve                1.8
spidev                   3.5
ssh-import-id            5.7
tensorboard              2.11.2
tensorboard-data-server  0.6.1
tensorboard-plugin-wit   1.8.1
tensorflow               2.4.0
tensorflow-estimator     2.4.0
termcolor                1.1.0
threadpoolctl            3.1.0
touchphat                0.0.1
twython                  3.7.0
typed-ast                1.3.1
typing-extensions        3.7.4.3
unicornhathd             0.0.4
urllib3                  1.24.1
virtualenv               20.23.0
webencodings             0.5.1
Werkzeug                 2.2.3
wheel                    0.40.0
wrapt                    1.12.1
zipp                     3.15.0
```

Figure 15: Packages installed on Raspberry Pi 3

d) **TensorFlow:** Installed according to the shell script from TensorFlow Github Package [21] there was a conflict with the OS settings when attempting to install the package. In order to resolve this issue, it is recommended to either downgrade the Python version or install a 64-bit version of the Raspberry Pi OS. For this project, the decision was made to downgrade the Python version to 3.7.12. We didn't install Keras as it's already available on TensorFlow.

e) **OpenCV:** The following resources are followed for installing OpenCV on the Raspberry Pi: a documentation [22] it should be mentioned that this package has numerous dependencies, which will prolong installation time.

f) **Matplotlib and sklearn:** Matplotlib and scikit-learn (sklearn) are Python libraries that are frequently utilized in data science and machine learning projects, particularly those related to TensorFlow library. In this project, these packages are downgraded and upgraded several times until they are compatible with the TensorFlow version.

g) **MediaPipe:** To successfully import this package, it is important to ensure that all dependencies are installed without conflicting with the OS release. In this project, the

OS release was Raspbian Bullseye (version 11). Additionally, compatibility with OpenCV was ensured to prevent any potential conflicts. The MediaPipe library was implemented by cloning its dependencies from the website for the library's official repository [15]. Notably sometimes importing MediaPipe works but could not functionalize. This could occur as a result of a conflict between the OS release requirements.

4.2.3 Space and Package Versions:

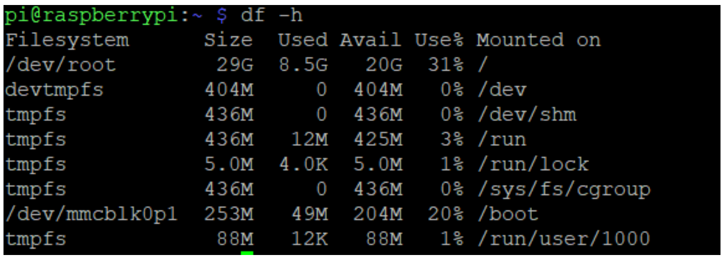
After installing all essential packages, regardless of the Mediapipe's inability to work, the consumed space memory was around 10 GB, which is 35% of the Micro SD card 32 GB. Below is a table showing how much space the card is using, along with another table revealing the necessarily packages that proposed to use for this ML project with their version:

Table 1: Size before ML Model dependencies

Filesystem	Size	Used	Avail	Use%
/dev/root	29G	9.6G	19G	35%

Table 2: Versions for Packages in Raspberry Pi 3

	picamera	TensorFlow	OpenCV	matplotlib	scikit-learn	NumPy	Mediapipe
version	1.13	2.4.0	4.7.0.72	3.5.0	0.24.2	1.21.6	0.8.13



```
pi@raspberrypi:~ $ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root        29G  8.5G   20G  31% /
devtmpfs        404M     0  404M   0% /dev
tmpfs           436M     0  436M   0% /dev/shm
tmpfs           436M   12M  425M   3% /run
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           436M     0  436M   0% /sys/fs/cgroup
/dev/mmcblk0p1  253M   49M  204M  20% /boot
tmpfs            88M   12K   88M   1% /run/user/1000
```

Figure 16: Space utilized on Raspberry Pi

4.2.4 Challenges and Outcome:

Implementing Sign Language Gesture Recognition using Computer Vision in real-time on a Raspberry Pi 3 with Raspbian Bullseye (version 11) presents a number of difficulties that prevent the project from being completed. They are briefly presented below:

- **Hardware and software compatibility:** Ensuring compatibility between hardware and software components can be challenging and may require additional configuration and troubleshooting. During the installation of Jupyter Notebook on a Raspberry Pi, we encountered a connectivity issue where the SSH connection was dropped every five minutes. After investigating the problem, we discovered that Jupyter Notebook uses a considerable amount of processing power and memory on the Raspberry Pi, leading to performance concerns. To address the Raspberry Pi's limited processing power, we recommend using external GPU accelerators to improve prediction and training speeds. One example of an external GPU accelerator that can be used is the Coral accelerator. By using an external accelerator, the Raspberry Pi can offload some of the processing tasks, leading to faster and more efficient machine learning tasks.

- **Dataset availability and technical limitations:** The Raspberry Pi's limited processor and memory capabilities can present challenges when working with large datasets for training and testing. Although we were able to capture our datasets (Figure 17), we experienced lag during the training period due to GPU compatibility issues. This highlights the limitations of using the Raspberry Pi 3 for complex machine learning tasks that require significant processing power and memory resources.

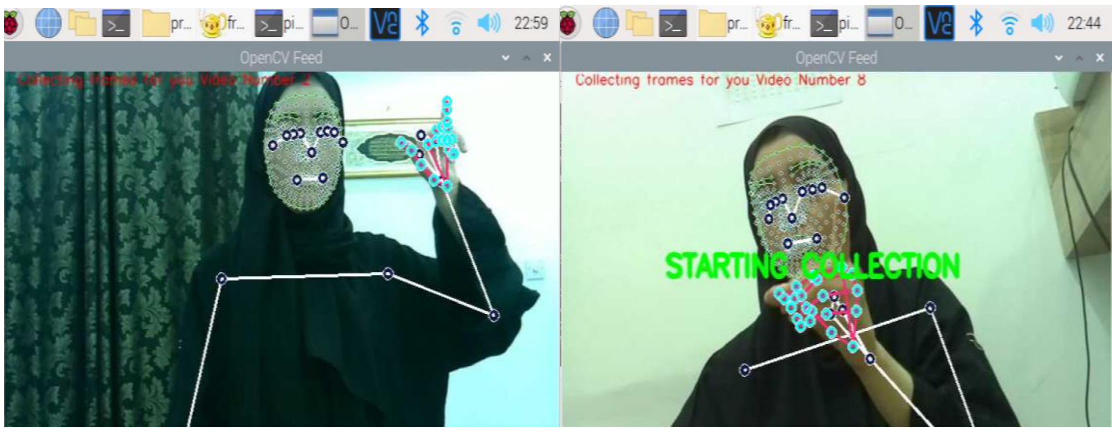

22:59
OpenCV Feed
Collecting frames for you Video Number 2
22:44
OpenCV Feed
Collecting frames for you Video Number 8
STARTING COLLECTION

Figure 17: Collecting SLGR Datasets on Raspberry Pi

The Hardware Implementation of our GSLR Model is shown in Figure 18. We remotely connected to our Raspberry Pi using VNC Viewer and connection was done using LAN

Wire. The Data collection and Real-time Prediction using Raspberry Pi was possible due to PiCamera. The Existing GPU is suggested to accelerate using an external TPU.

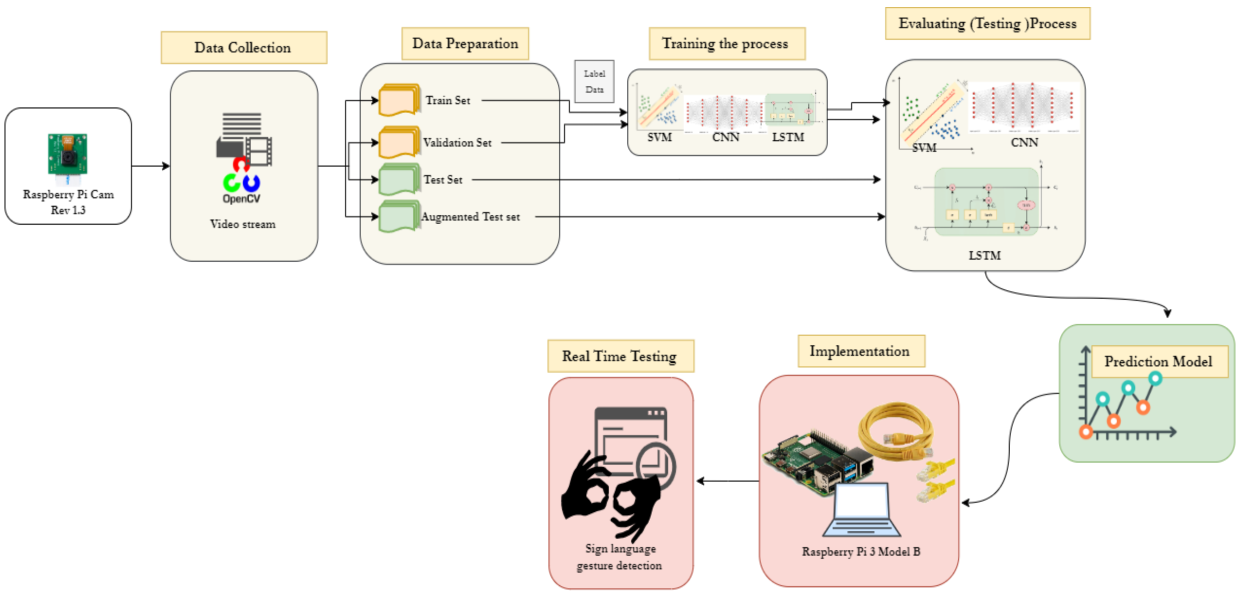


Figure 18: Hardware Implementation of GSLR Model

By using VNC Viewer and a LAN wire, we could remotely connect to the Raspberry Pi from another device. This allowed us to access the Raspberry Pi's desktop environment and run the GSLR model as if we were physically sitting in front of the Raspberry Pi. We suggest using an external TPU, it can further accelerate the computations required by the GSLR model. This can result in even faster and more efficient processing of data.

Given the time constraints and challenges associated with the development of the Raspberry Pi system, addressing these issues has been identified as a priority for future work.

Chapter 5
Testing and Results

This chapter provides the results of ML training models that have been done and the model's classification performance in terms of confusion for each trained model. In addition to the models training and testing accuracy regarding SL actions, video frames, testing size and hyperparameters.

5.1 Model Performance

Machine learning (ML) model performance is typically evaluated using various metrics, depending on the problem being addressed. Some common metrics for evaluating our ML model performance includes Accuracy and Confusion Matrix.

A confusion matrix (Figure 19) can indicate training performance by providing a detailed breakdown of the model's predictions across different classes in the training data. A confusion matrix is a square matrix that displays the number of true positives, true negatives, false positives, and false negatives for each class in the training data.

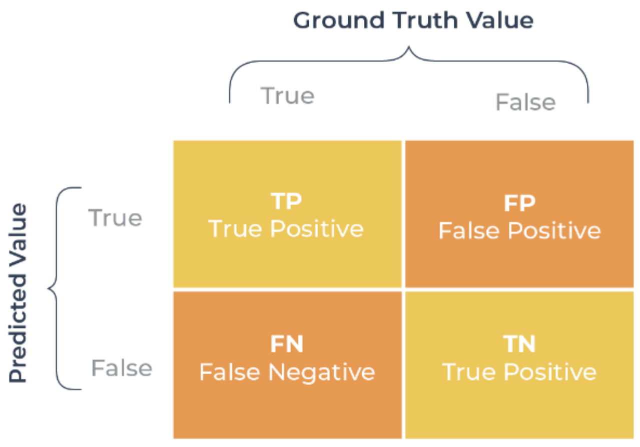


Figure 19: An overview of Confusion Matrix

The true positive (TP) and true negative (TN) values represent the number of correct predictions made by the model, while the false positive (FP) and false negative (FN) values represent incorrect predictions. The confusion matrix can be used to calculate various performance metrics, including accuracy, precision, recall, and F1 score.

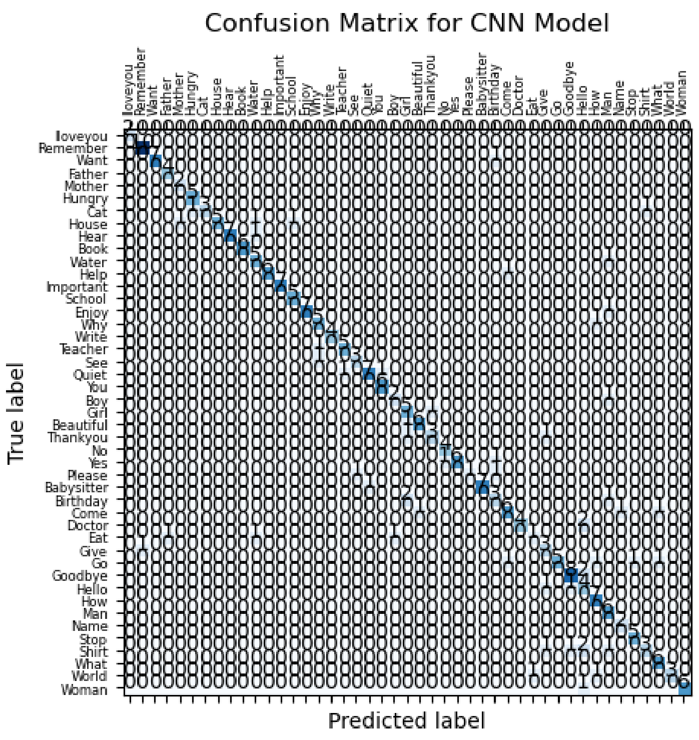


Figure 20: Confusion Matrix for CNN Model on 45 Word-level SL Gestures

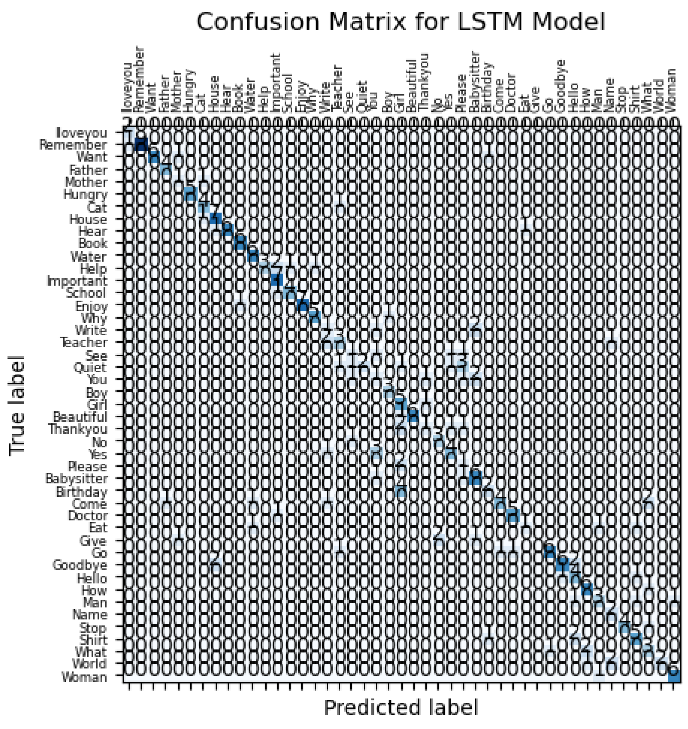


Figure 21: Confusion Matrix for LSTM Model on 45 Word-level SL Gestures

Overall, a confusion matrix can provide valuable insights into the training performance of a model and can be used to identify areas for improvement in the model architecture or training data.

To obtain further accuracy the system we suggest implemented the system using “WLASL: A large-scale dataset for Word-Level American Sign Language” [18] which features 2,000 common different words in ASL.

5.2 Predictions for ML Models

During our initial experimental phases, we trained our model using only 5 words to evaluate our prediction accuracy and developed a user visualization plan for these 5 words. However, we soon realized that adding more words to our model would make it more complex and difficult to understand and would go beyond the scope of our initial proposal (Figure 22).

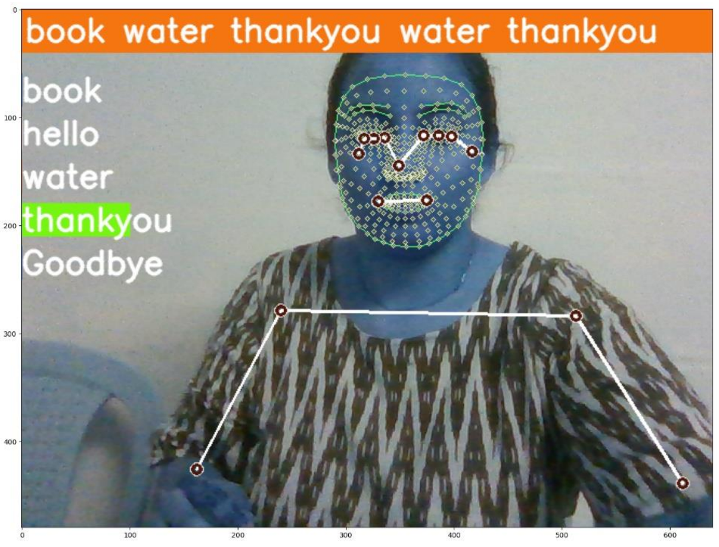
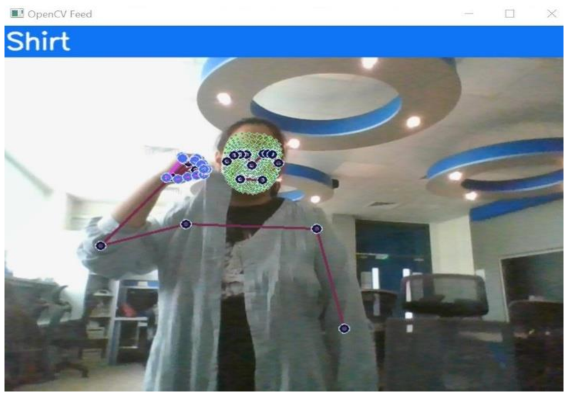
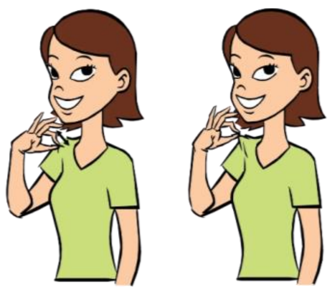
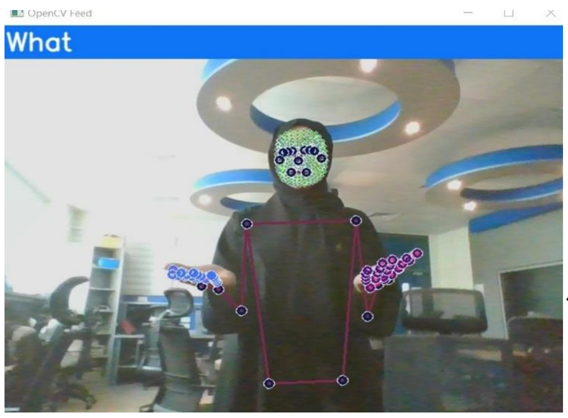
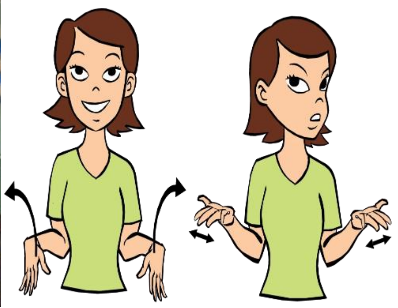
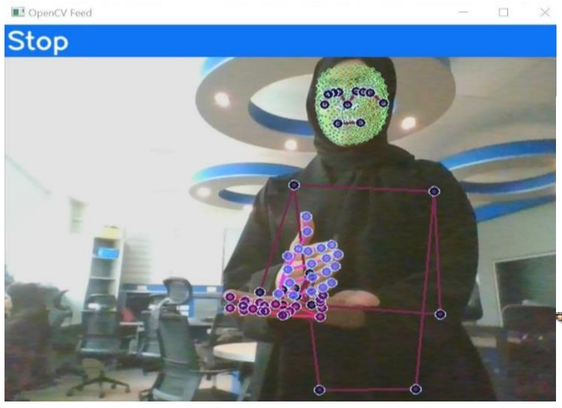


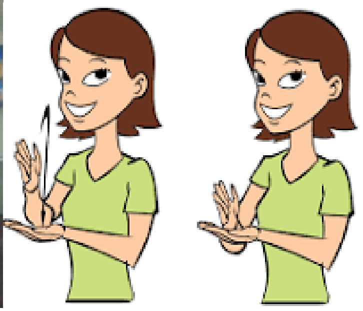
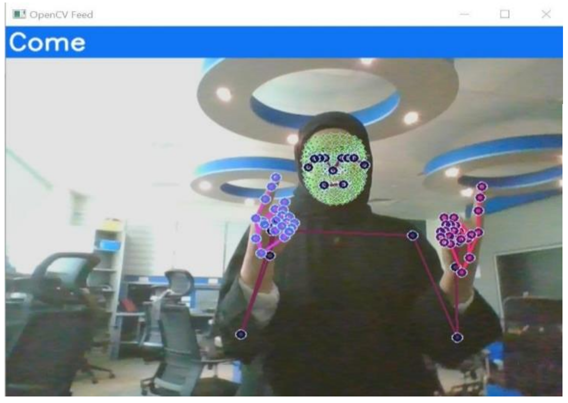
Figure 22: Initial Visual Prediction of LSTM Model with 5 words

In our SLGR project, we rely heavily on Mediapipe keypoints during the training process. To ensure that our model accurately predicts SL gestures in real-time, we use visual prediction techniques such as SL images to verify the accuracy of the predicted gestures. This involves comparing the predicted gestures and keypoints to the actual ones in the SL images as in Figure 23 and 24 (courtesy: BabySignLanguage), allowing us to confirm that the model is correctly identifying and predicting SL gestures.



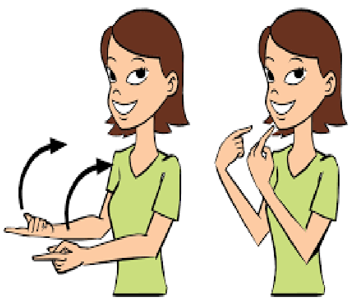


Figure 23: Final Prediction real-time for SL Gestures Shirt, What, Stop, Come

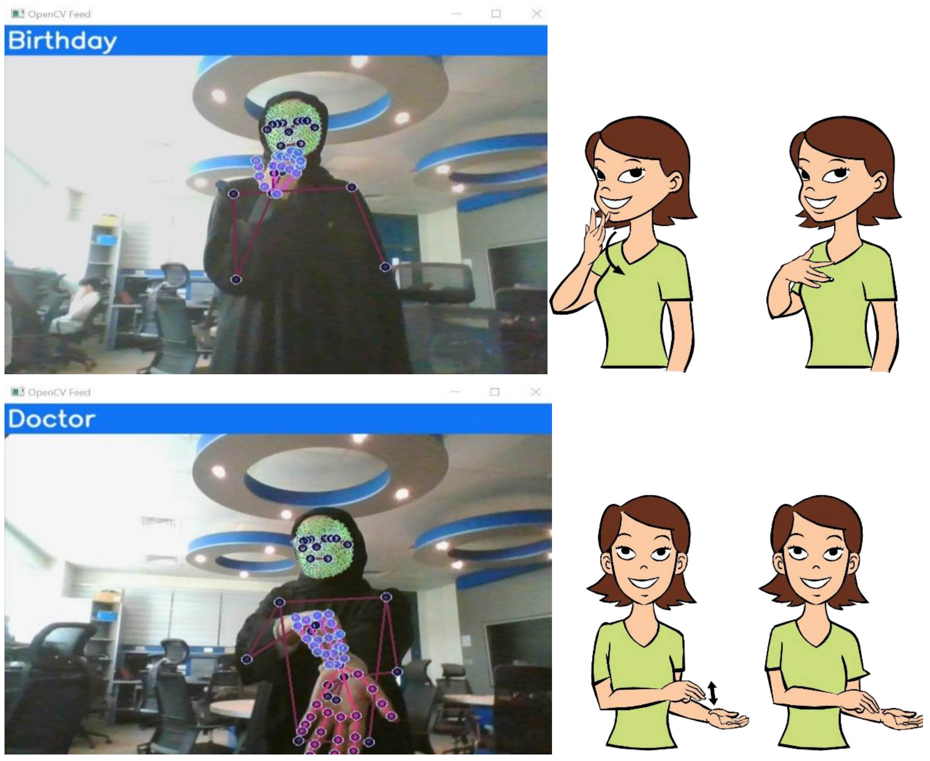


Figure 24: Final Prediction on Real-time for Birthday and Doctor

During our real-time prediction in the SLGR project, we observed that there was some lag time during the initial system load. Additionally, we faced some delays and lags in gesture response when using the CNN model for prediction.

We also encountered some challenges with misinterpretation of action gestures for words with remarkably similar dynamic motion, such as "come" and "go", "woman" and "mother", and "man" and "father". These similarities in motion resulted in the model misinterpreting the gestures, leading to errors in prediction.

Furthermore, we experienced difficulties with gestures that had multiple poses, such as complex signs or phrases. Our LSTM system required multiple trials to detect and interpret these gestures accurately.

5.3 Model Comparison

During our project we understand it's important to consider the trade-offs between model complexity, training time, and performance when selecting the best model for a given task (detailed in section 5.2). A more complex model may achieve higher accuracy but may also require more training time and computational resources. Comparing accuracy is a common way to evaluate to different machine learning (ML) models, but it's important to consider the limitations of this approach. We have provided our experimental accuracy stages to validate our decision on final model building and training as shown in Table 3 maximum accuracy achieved by the LSTM model at the beginning of the training process was less than 80% (see section 5.2).

Table 3: Experimental ML Model Accuracy Results

Experimental ML Model Accuracy		
Dataset	Model	Accuracy
5 Word-level SL Gestures 15 frames, Test size = 0.2	LSTM (3-Layer)	Train: 78.94% Test: 60%
16 Word-level SL Gestures 30 frames, Test size = 0.05	LSTM (4-Layer)	Train: 81% Test: 66.67%
16 Word-level SL Gestures 30 frames, Test size = 0.2	LSTM (3-Layer)	Train: 86% Test: 75%
47 Word-level SL Gestures 30 frames, Test size = 0.05	LSTM	Train: 80.56% Test: 75.93%

Based on the model comparison performed on Table 4, the CNN model had the highest accuracy on the ASL dataset, with an accuracy of over 95%. This is a promising result, as it indicates that the CNN model accurately classified the ASL gestures with high accuracy.

Table 4: Final SLGR Model Accuracy Results

Final ML Model Accuracy			
Dataset	Model	Accuracy	Time
45 word-level SL Gestures 30 frames Test size = 0.2	SVM	Train: 87.59% Test: 74.44%	Train: 69.30 seconds Test: 16.73 seconds
	CNN (3-Layer)	Train: 98.15% Test: 80.37%	Train: 584.44 seconds Test: 2.71 seconds

	LSTM	Train: 80.56% Test: 65.93%	Train: 82.29 seconds Test: 0.61 seconds

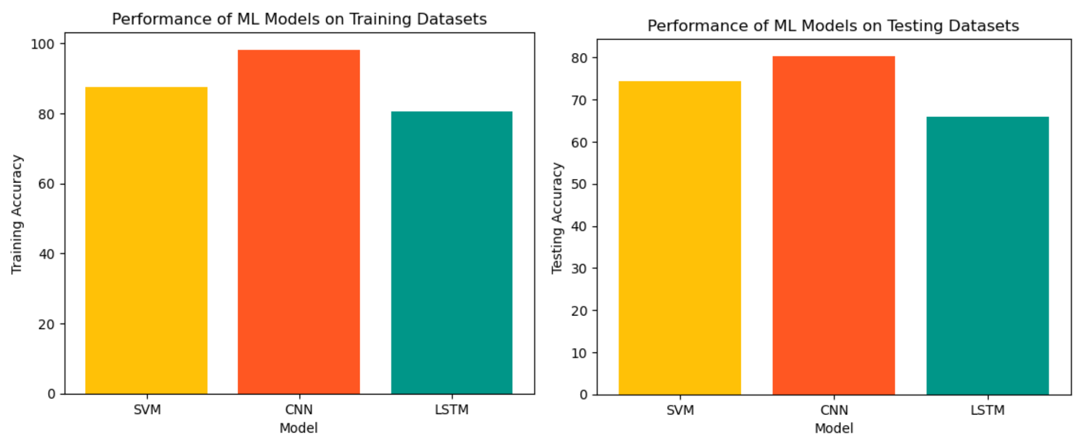


Figure 25: Graph Comparing the performance of SVM, LSTM and CNN

Additionally, accuracy may not always reflect the true performance of the model in real-world settings, as it may be influenced by factors such as the quality and size of the training data, the complexity of the model, or the choice of hyperparameters.

We also conducted an experiment to assess the impact of image quality on the accuracy of your SLGR model. Specifically, we captured the same SL dataset using both an RGB camera and web camera and compared the accuracy and prediction results of our model using both sets of images.Based on our results, we found that the image quality had a negligible impact on the accuracy of your model, with the RGB images providing comparable results to the other type of camera. This is a useful finding, as it suggests that using lower quality images may not significantly affect the accuracy and reliability of your SL recognition

Chapter 6
Conclusion and Future Work

Our project is intended to create intelligent interfaces to improve communication by merging sign language recognition, visual idea identification, and natural language-generating technologies. Furthermore, the technology will aid in the automatic production of text/speech translations to improve accessibility and inclusion. Additionally, contribute to the advancement of computer vision and sign language recognition to improve accessibility and inclusion in society. Moreover, the models were trained with varying quantities of sequences, frames, and words to analyze how these variables impact model accuracy and identify which model achieved the best accuracy. Comparing accuracy is a typical approach for evaluating and comparing the performance of several machine learning models by considering the trade-offs between model complexity, training time, and performance when picking the optimal model for a particular job. The CNN model produced the most accurate results in all studies, with an accuracy of more than 95% on the ASL dataset. This is an encouraging finding since it demonstrates that the CNN model correctly categorized the sign language movements.

The scope of our project was limited in terms of prediction as our model takes a sequence of data rather than a single frame for detection which eradicates the chances to produce wrong predictions. Unfortunately, due to a lack of prior expertise and time restrictions, it was only implemented with the Raspberry Pi 3. Furthermore, the number of layers in our study was limited owing to computing capacity and training resources such as high-end CPUs, GPUs, or specialized hardware such as Tensor Processing Units (TPUs). Despite the lack of big datasets and the capacity to detect complex behaviors across different languages, suggested models often focus on single languages, such as American Sign Language (ASL), to overcome these issues.

The first potential future work area is to complete deployment of the machine learning model in Raspberry Pi, addressing the basic guides mentioned in Raspberry Pi implementation chapter. Another future work for research purposes is to investigate the prediction of implementing sign language recognition multi-modal approaches for various other Sign Languages and to distinguish between multi-model combinations predictions. The addition of error management to the sign language recognition system is also another area that needs consideration for future work. This would involve implementing try-except blocks to handle potential failures, such as camera disconnections, errors in loading the model, or unexpected inputs as well as improvement on final visual screen. By adding error handling, the system can become more stable and reliable in handling unexpected situations. Furthermore, it will be an effective approach for the future work to permit dynamic thresholding. Allowing for dynamic thresholding, in which the threshold is adjusted based on the current probability distribution, may increase the accurate results. Additionally, recognizing signs in a continuous stream of signing differing across various locations and countries, which more closely simulates natural sign language communication, is a recommended practice for project enhancement. Due to time and resource limitations, building and training a hybrid model for Sign Language Gesture Recognition was not possible. However, these tasks can be considered for future work with additional research and development. If successful, the system has the potential to revolutionize communication for individuals with hearing impairments across various settings, including education, healthcare, entertainment, and social engagement. This could significantly improve their quality of life and enhance their ability to participate in society.

References

List of References:

[1] "Deafness and hearing loss," World Health Organization, [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss . [Accessed: May 6, 2023].

[2] McCaskill, C., Lucas, C., Bayley, R., and Hill, J. "The hidden treasure of Black ASL: Its history and structure." Gallaudet University Press, Washington, DC, 2011.

[3] K. Murakami and H. Taguchi, "Gesture recognition using recurrent neural networks," in Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems, pp. 237-242, 1991, doi: 10.1145/1088444.108900.

[4] O. M. Sincan, and H. Y. Keles, "Autsl: A large scale multi-modal Turkish sign language dataset and baseline methods," IEEE Access, vol. 8, pp. 181340-181355, 2020, doi: 10.1109/ACCESS.2020.3020214.

[5] R. Y. Wang and J. Popović, "Real-time hand-tracking with a color glove," ACM Transactions on Graphics (TOG), vol. 28, no. 3, p. 63, 2009, doi: 10.1145/1531326.1531381.

[6] R. Sharma, R. Khapra, and N. Dahiya, "Sign Language Gesture Recognition," pp. 14-19, June 2020, doi: 10.1109/ICBSP51964.2020.9213113.

[7] A. Halder and A. Tayade, "Real-time vernacular sign language recognition using mediapipe and machine learning," International Journal of Recent Technology and Engineering (IJRTE), vol. 10, no. 3, pp. 7421-7423, 2021, ISSN: 2277-3878, doi: 10.35940/ijrte.C1193.1183S319.

[8] W. Liu, Y. Fan, Z. Li, and Z. Zhang, "RGBD video based human hand trajectory tracking and gesture recognition system," Mathematical Problems in Engineering, vol. 2015, Jan. 2015, doi: 10.1155/2015/954631.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, June 2016, pp. 779-788.

[10] Y. Song, J. Yuan, and X. Li, "Real-time hand gesture recognition using YOLO architecture," IEEE Access, vol. 8, pp. 63752-63759, 2020.

[11] Y. Wang, G. Zhang, Y. Jiang, and D. Wu, "Sign Language Action Detection based on Multi-branch Convolutional Neural Network with Spatiotemporal Attention Mechanism," in Proceedings of the 2021 IEEE International Conference on Image Processing (ICIP), 2021, pp. 2111-2115.

[12] Google Inc., "MediaPipe Face Mesh," GitHub, 2021. [Online]. Available: https://github.com/google/mediapipe/wiki/MediaPipe-Face-Mes h. [Accessed: May 9, 2023].

[13] J.W. Kim, J.-Y. Choi, E.-J. Ha, and J.-H. Choi, “Human Pose Estimation Using MediaPipe Pose and Optimization Method Based on a Humanoid Model,” Applied Sciences, vol. 13, no. 4, p. 2700, Feb. 2023, doi: 10.3390/app13042700 .

[14] Google. "MediaPipe Gesture Recognizer." MediaPipe Solutions: Vision, Google Developers, 2021, https://developers.google.com/mediapipe/solutions/vision/gesture_recognizer .

[15] Google. "MediaPipe Holistic." GitHub, 2021, https://github.com/google/mediapipe/blob/master/docs/solutions/holistic.md .

[16] T. Ray, "What and why behind fit_transform() and transform() | Towards Data ..." Towards Data Science, 2019. [Online]. Available: https://towardsdatascience.com/what-and-why-behind-fit-transform-vs-transform-in-scikit-learn-78f915cf96fe . [Accessed: May 14, 2023].

[17] "Raspberry Pi 3 Model B WiFi & Bluetooth Setup - Device Plus," Device Plus, [Online]. Available: https://www.deviceplus.com/raspberry-pi/raspberry-pi-3-model-b-wifi-bluetooth-setup/ . [Accessed: May 15, 2023].

[18] Li, Du, et al. "WLASL: A Large-Scale Dataset for Word-Level American Sign Language." WLASL Dataset, 2021, https://dxli94.github.io/WLASL/.

[19] Raspbian. (n.d.). Raspbian Mirrors. [Online]. Available: http://raspbian.org/RaspbianMirrors. [Accessed: May 14, 2023].

[20] Pi Supply. (n.d.). Raspberry Pi Camera Board V1.3 - 5MP 1080p. [Online]. Available: https://uk.pi-supply.com/products/raspberry-pi-camera-board-v1-3-5mp-1080p. [Accessed: May 16, 2023]

[21] PINTO0309 TensorFlow-bin/previous_versions. [Online]. Available: https://github.com/PINTO0309/Tensorflow-bin/tree/main/previous_versions. [Accessed: May 14, 2023].

[22] OpenCV: Install OpenCV-Python in Ubuntu." [Online]. Available: https://docs.opencv.org/4.x/d2/de6/tutorial_py_setup_in_ubuntu.html. [Accessed: May 12, 2023].