ONLINE LEARNING PLATFORM FOR HEARING IMPAIRED PEOPLE

Project ID: 2022-59

Final Report

Ramawickrama H.N – IT19174686

Bachelor of Science (Hons) Degree in Information Technology

Specializing in Data Science

Department of Information Technology

Faculty of Computing

Sri Lanka Institute of Information Technology

Sri Lanka

March 2022

ONLINE LEARNING PLATFORM FOR HEARING IMPAIRED PEOPLE

Project ID: 2022-59

Final Report

Ramawickrama H.N – IT19174686

Supervised by – Dr. Lakmini Abeywardhana

Bachelor of Science (Hons) Degree in Information Technology

Specializing in Data Science

Department of Information Technology

Faculty of Computing

Sri Lanka Institute of Information Technology

Sri Lanka

March 2022

DECLARATION

We declare that this is our own work, and this proposal does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any other university or Institute of higher learning, and to the best of our knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgment is made in the text.

Name	Student ID	Signature
Ramawickrama H.N.	IT19174686	[illegible]

The above candidates are carrying out research for the undergraduate Dissertation under my supervision.

……………………………. …………………

Signature of the supervisor Date

(Dr. Lakmini Abeywardhana)

…………………………….. ………………….

Signature of the Co-supervisor Date

(Mr.YashasMallawarachchi)

ABSTRACT

Communication is the masterpiece of human interaction in society, out of which, sign language, which is a gestural language has become the primary means of communication of the deaf community worldwide. The majority of children with hearing loss suffer from various stigmas such as social isolation, loneliness, and anxiety due to lack of communication. Considering this matter many approaches are being made to reduce the communication gap between the deaf and the ordinary communities. Nevertheless, there are 138 to 300 varieties of sign languages across the world, there is no universal sign language that is being standardized. This has caused the HIC to be unaware of a foreign sign language, creating destruction for communication in a preferred language of their own. Through this research component, we mainly focus on the language barrier within the HIC, where both terminals have difficulties in communication due to the absence of knowledge of each other's sign language. As a result, this research gap covers the necessity of a system that translates Sri Lankan sign language into American Sign language (ASL). This enables the Sri Lankan hearing impaired community to interact with other similar users across the world by learning the translated sign language and using it in communication. The proposed system will use computer vision and machine learning techniques to capture the hand gestures through a webcam and convert them to a text format, which will then be translated to ASL. Results will be presented through a 3D avatar model.

Keywords: Sri Lankan Sign language, American Sign Language, Hearing Impaired Community, Computer Vision

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to our supervisor Dr. Lakmini Abeywardhana for providing invaluable guidance and support throughout this research. I would also like to extend my gratitude to our co-supervisor Mr. Yashas Mallawarachchi for the assistance that was provided to make this project a success.

I am truly grateful to my family, my group members, and the former lecturers for providing immense support as the completion of this project would not have been possible without their assistance. Finally of all those whose names are not mentioned in particular, I am truly thankful for the support that I have received from all of them to bring this project to an end.

TABLE OF CONTENT

TABLE OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATION

SSL	SriLankan Sign Language
BSL	British Sign Language
ASL	American Sign Language
OC	Ordinary Community
HIC	Hearing Impaired Community
SL	Sign Language
ML	Machine Learning
CNN	Convolutional Neural Network

1 INTRODUCTION

1.1 Background Study

Sign language is a gestural language used by the HIC through bodily movements as their mode of communication [1]. There are around 138 to 300 different sign languages around the world. However, there is no universally accepted sign language that is being standardized yet [2]. The countries that share the same spoken language do not share the same sign language. When considering English, it has three varieties of sign languages: American sign language (ASL), British sign language (BSL), and Australian sign language [3] each of which is different from the other. The figure below that was taken from a source, displays how various sign languages distribute across the world [4].

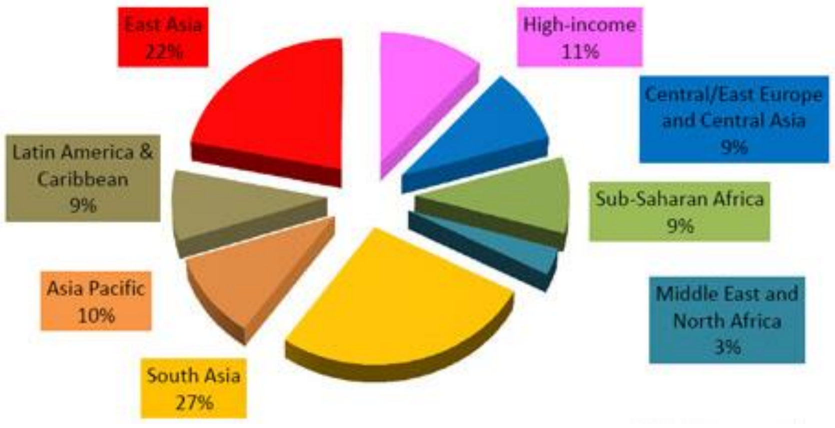


Figure 1 Sign language distribution across the world

Since the ordinary community does not understand sign language and due to the fact that there is no standardized sign language for the hearing-impaired community to engage worldwide, there have been many problems have arisen. Most of which are problems such as social isolation, loneliness, or anxiety disorders that are commonly seen within the HIC [5].

With the evolving technology, numerous chat applications and visual communication channels are being introduced to interact even with someone who is in a different geographical location. In such situations, the HIC may be deprived of opportunities due to the absence of knowledge of another foreign sign language [6].

An article released by the Australian Department of social services, reports on the supply and demand for interpreters for the deaf community in Australia [7]. According to the article, it's reported that many deaf users rely on interpreters in their day-to-day interactions. Figure 1.1.2 below which was taken from a source, depicts how often the HIC need interpretations during their regular daily activities in Australia.

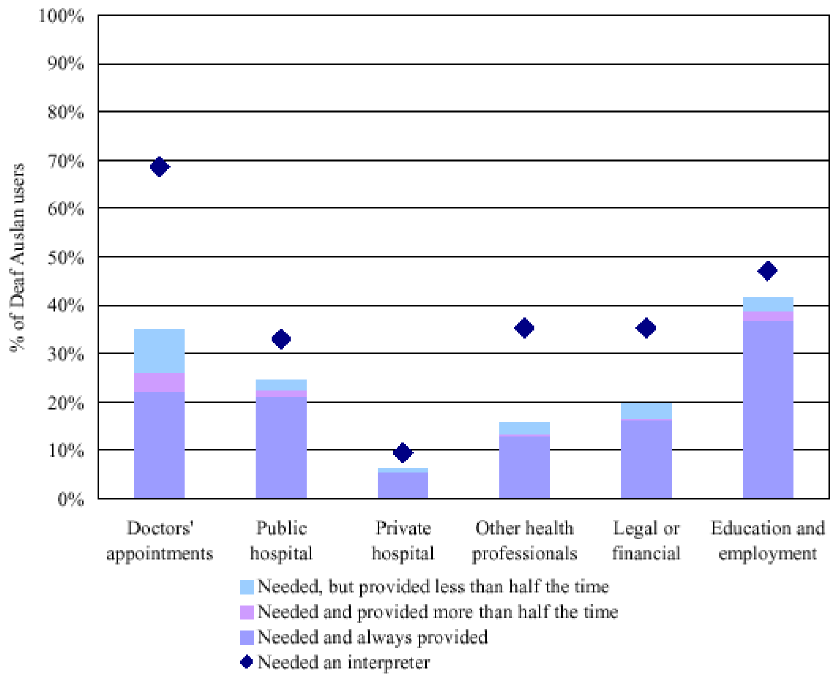


Figure 2 Need for interpreters for the HIC in Australia

This can lead us to make an assumption that developing countries like Sri Lanka which lacks most of the resources and facilities, would have a higher need for interpreters in such situations.

According to an article in ‘Conversation’, there are various approaches made by different countries to address hearing loss which includes cochlear implants, hearing aids, and learning sign language [8]. But it is not always possible for the HIC to access them due to economic limitations and lack of resources in many countries.
As a result, this approach is made to develop a system that can translate between sign languages in order to address the above-mentioned problems and give more exposure to the HIC to learn a non-native sign language.

1.2 Literature Survey

According to the world health organization more than 430 million people suffer from hearing loss [1]. The article ‘verywellmind’ states that a large part of the HIC suffers from various psychiatric conditions such as mood disorders which are mainly due to communication difficulties [9]. This primarily includes lack of interpretation between sign language and spoken language. As a result, the need for a suitable system that can help reduce the communication barrier within these communities has come to the attention of many researchers. Hence, research is conducted in order to enhance the opportunities for the HIC to interact efficiently with one another.
The mobile-based application ‘'Sanwadha” provides real-time communication between the ordinary and the hearing-impaired community [10]. The application consists of functionalities such as the conversion of text into Sign language, voice to sign language, and GIF conversion. In text conversion, once a text is entered in English or Sinhala, the set of strings will be translated into sign language and will get transformed into a GIF format. Similarly in voice conversion, the application introduces a 2D model or an animated sticker for the deaf user to input the sign and the result will be delivered to the normal user through a voice output.
EasyTalk is another similar application that translates SSL into text and audio formats as well as verbal language into the SSL [11].

This application captures the hand signs through a hand gesture detector using pre-trained models and using an image classifying component it classifies and translates the detected hand signs. The identified hand signs produce a text or an audio formatted output with the aid of text and voice generator components.

UTalk is an SSL converter developed using Computer vision and Machine learning techniques [12]. This mobile application focuses on interpreting static as well as dynamic signs that are expressed in SSL. The system captures the video of the user performing sign language as the input while extracting the frame segment and removing the background out of those frames with the aid of image processing techniques. These frames that are being preprocessed are made to go through two separate machine learning models stated as static sign classifier and dynamic sign classifier, after which the output will be fed to a language model and presented in a text format.

Another research introduced static sign language recognition using deep learning [13]. The system was based on a skin color modeling technique where the predefined skin color range will extract the pixels from non-pixels, in other words, the hand is separately recognized from the background. The images are then fed into the model Convolutional Neural Network (CNN) for image classification and are later trained using Keras. The system obtained an accuracy of 93.67%

A Japanese team of researchers developed a CG-Animation system for sign language communication between different languages [14]. The system analyzes the image of the sign language gesture, and the image is described through text and programs. CG animation is generated through these texts and programs. Apart from sign language to text conversion, it's been noted that there are a few other systems that have implemented text conversion into the relevant sign language.

Considering text-to-sign language conversion, a study was conducted by a group of researchers from the University of Pennsylvania where they implemented a system that converts English to ASL [15]. The implementation is done based on two approaches where initially the English text which is taken as the input is converted into an

intermediate representation after which it is further converted to a set of quantitative parameters which control an articulated 3D human model to produce the ASL.
Similar research was done using Russian sign language. In this research, a semantic analysis algorithm is developed and introduced. The aim of semantic analysis is to model the meaning of the words in the sentence [16].

1.3 Research Gap

As mentioned in the above literature survey many research projects are proposed to interpret sign language into spoken language or vice versa. However due to the fact that there is no universal sign language for the HIC across the world, and different countries have their own unique sign language, a system that is implemented for one particular sign language cannot be directly used to understand another. Hence the necessity for a platform that can translate one sign language to another remains yet to be introduced. Considering these issues in the existing systems, we have presented a solution in the proposed system to convert SSL to ASL. Shown below is a summary of some of the notable gaps that were found in the previous research projects. Table below will show a list of previous research compared with our proposed system.

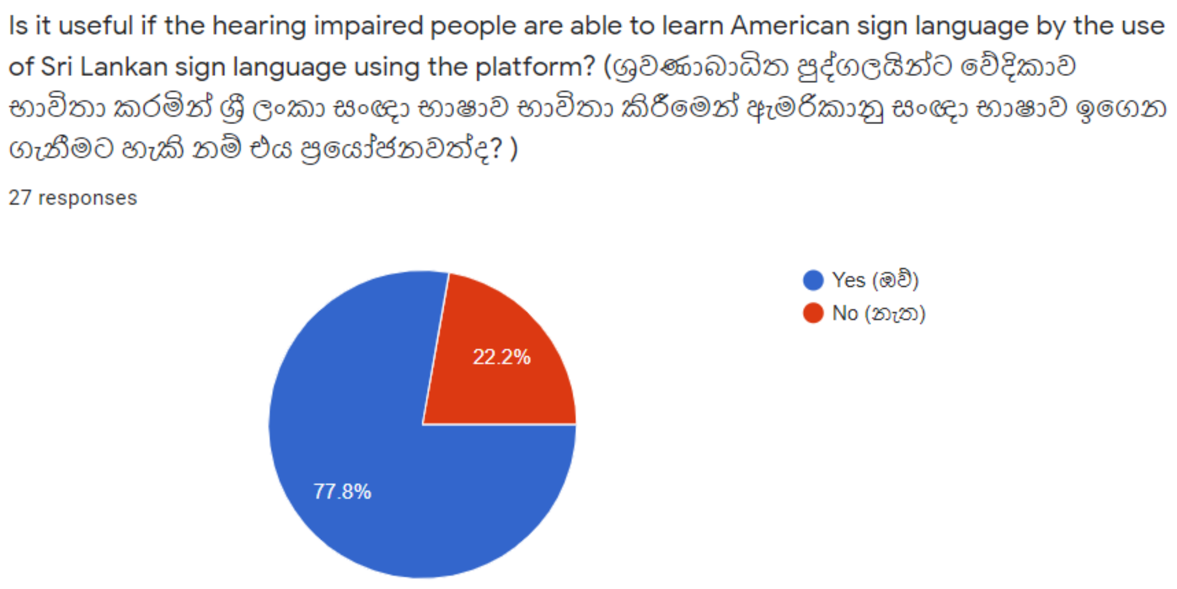
Table 1 Comparison of the current research and previous approaches

	Conversion of natural language (text/audio to sign language)	Conversion of sign language to natural language (text/audio)	Translation of one sign language to another	Based on SSL	Computer Vision based approach
Utalk: Sri Lankan Sign Language Converter Mobile App	✗	✓	✗	✓	✓
Static Sign Language Recognition Using Deep Learning	✗	✓	✗	✗	✓
Sanwadha: Mobile Assistant for Hearing Impairers	✓	✗	✗	✓	✓
A Machine Translation System from English to American Sign Language	✓	✗	✗	✗	✓
EasyTalk: A Translator for Sri Lankan Sign Language	✓	✓	✗	✓	✓
Proposed System	✓	✓	✓	✓	✓

1.4 Research Problem

Most of the existing systems were implemented to facilitate the necessity to translate from a sign language to a spoken language or vice versa. However, a system that is implemented to understand one particular sign language cannot be used to understand another foreign sign language. Due to the unavailability of such interpreters, there is a need for a platform that can provide the HIC to familiarize themselves with various other sign languages. Therefore, this application reaches out to the HIC in Srilanka who needs support with getting SSL translated to a non-native sign language which would further enhance their knowledge of a sign language that they are not familiar with while expanding their communication skills.

A survey that was conducted with 27 participants presented that it’s beneficial to have a platform that can be used to learn the American Sign language. Figure below shows a diagram of responses to the conducted survey.



Is it useful if the hearing impaired people are able to learn American sign language by the use of Sri Lankan sign language using the platform? (ශ්‍රවණාබාධිත පුද්ගලයින්ට වේදිකාව භාවිතා කරමින් ශ්‍රී ලංකා සංඥා භාෂාව භාවිතා කිරීමෙන් ඇමරිකානු සංඥා භාෂාව ඉගෙන ගැනීමට හැකි නම් එය ප්‍රයෝජනවත්ද?)

27 responses

Figure 3 Survey Results

A 77.8% of a notable number of responses were given for the above questionnaire which indicates that this platform will be useful for a significant number of people. As a result the approach is carried out as a beginner level translating system that can translate some of the words that are used regularly in the day to day activities into ASL

2 OBJECTIVES

2.1 Main Objectives

Main objective of this research is to develop a system that can facilitate the HIC in SriLanka to learn ASL in an efficient manner and discover different aspects of sign language worldwide. The system focuses on a few of the basic categories of words that are used in daily activities. The main objective is to translate a given gesture in SSL and convert it into text so that the user can verify that the given gesture is recognized correctly. Subsequently, the recognized gesture will be translated into ASL denoting the hand sign with a pretrained avatar model.

2.2 Specific Objectives

Some of the specific objectives of the research within the main objectives are:

- Creating an interactive and efficient learning platform for the HIC by enhancing their communication aspects.
- Ensure that an accurate response is given by the system according to the user requirements.

3 METHODOLOGY

3.1 Methodology

3.1.1 System Architecture Diagram

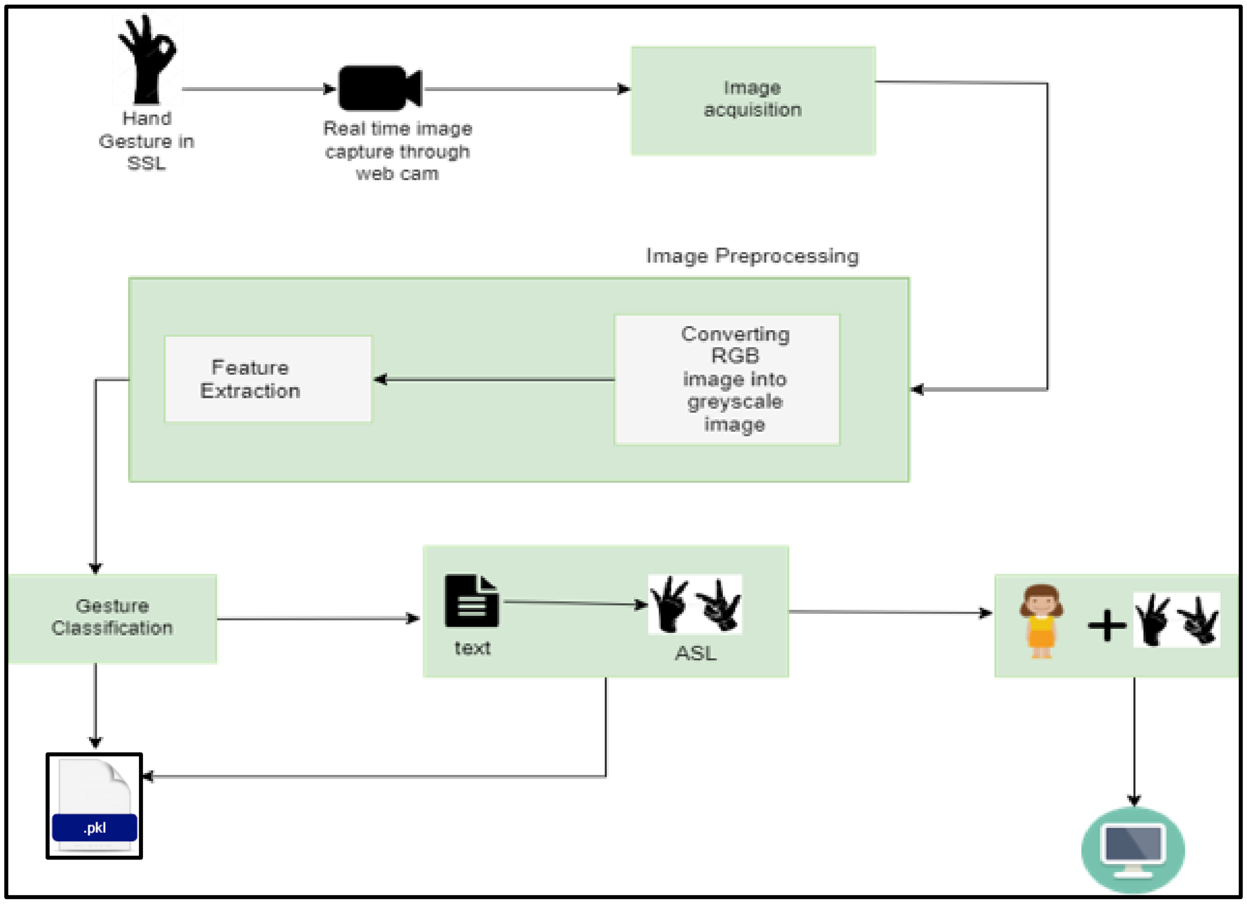


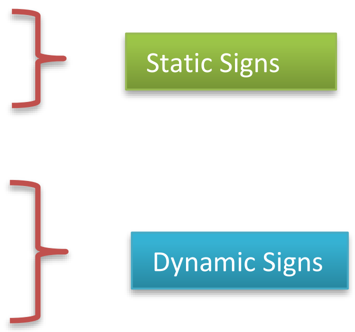
Figure 4 System Architecture Diagram

3.1.2 Conversion of SSL to text

As the first half of the component, a selected set of categories of gestures was chosen to convert into text. Considering the difficulties in finding datasets in Sri Lankan sign language, a dataset was generated by collecting images of the hand gestures from the web camera and passed to train. A collection of words were selected to translate initially belonging into the following categories:

- English Alphabet
- Numbers [1-10]

- Days in a week
- Months in a year
- Colours.



Considering sign language, gestures mainly fall under two main categories as static and dynamic signs. Therefore, the English alphabet and Numbers [1-10] which contains static gestures are trained using a CNN model through image processing while dynamic gestures belonging to Days of the week, Months, and Colours are trained with the use of an LSTM model through video processing.

When considering the static signs an image count of roughly around 300-400 was collected per class from the web camera, summing up to 5000+ images overall. Considering that the original images that were collected come in an RGB color format, they are being converted into a greyscale color composition. Through this color arrangement, the amount of data in a single image could be further reduced by displaying the image in black and white colors. This will eventually make it easier to recognize the gesture explicitly and enhance the accuracy of the overall system. The captured images were converted into a grayscale image and are further resized. The collected data is fed into a model for the training process. Upon obtaining a considerable accuracy the weights of the models are saved in a file. Based on the trained weights the detection will take place where the given gesture in SSL will be translated into a text form.

3.1.3 Conversion of text to ASL

The latter part of the component is to translate the converted text to the relevant American sign. The overall text results are saved into a pickle file and are mapped with the corresponding American gestures. The results will be sent through a Flask API. Moreover, an avatar model is trained to denote the translated words. Ultimately the user will have a button click option to select which category needs to be translated. After the selection, the relevant Sri Lankan sign can be given as a camera input for translation. The translated gesture will finally be displayed as an avatar to the user.

3.1.4 Training the avatar model

The avatar models were trained using ‘DeepMotion’ and ‘3D Hand Draw’ applications. DeepMotion is a motion tracking application where it tracks bodily movements including hands and facial gestures. When a video clip of performing gesture was provided to the application, a prebuilt avatar model imitates the movements of the given video. The avatar movements were then recorded and taken into small video clips for each relevant gesture.

3D Hand Draw application was used to operate the hand movements of the gestures. Therefore, this was mainly used for the sign language gestures that were only indicated using hand movements. A pre-generated avatar was provided by the application. Using the provided avatar model small video clips were obtained by operating its’ hand and finger movements.

3.2 Implementation

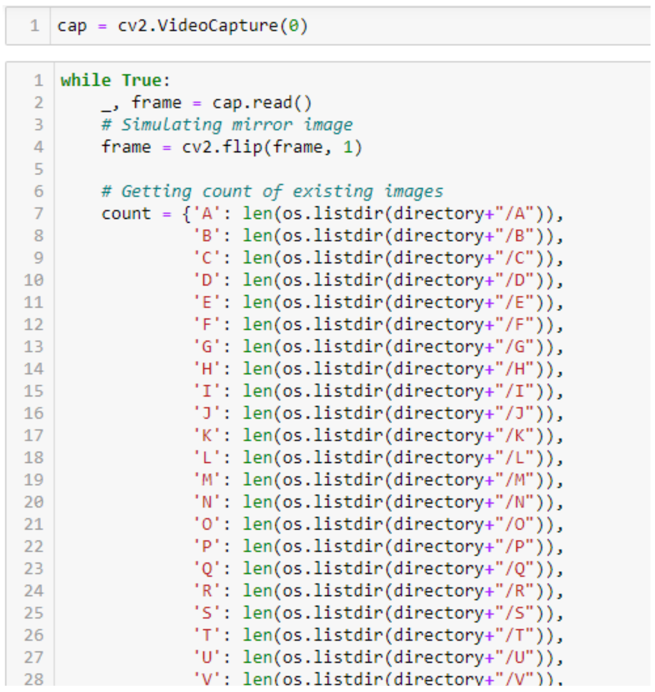
Pre-Requisites:

- Visual Studio Code
- Jupyter Notebook

3.2.1 Data Collection

The data set was created by gathering images from the web camera using OpenCV. It was collected for 4 different categories of words using images and videos based on whether the gesture is static or dynamic.

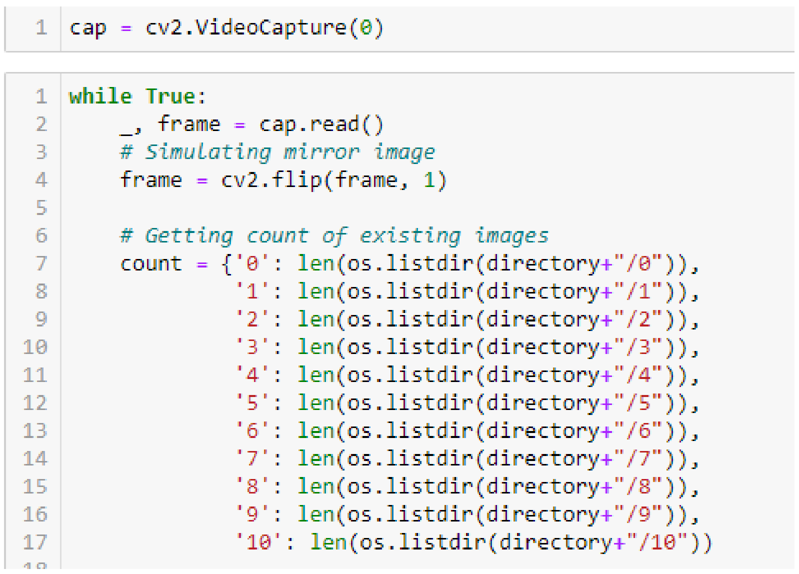
For static gesture categories such as the Alphabet and Numbers, a sequence of images were collected. An image count of 200-300 images was obtained for each class and was sent into the relevant directories upon capturing.



```python
cap = cv2.VideoCapture(0)
```

```python
while True:
    _, frame = cap.read()
    # Simulating mirror image
    frame = cv2.flip(frame, 1)

    # Getting count of existing images
    count = {'A': len(os.listdir(directory+"/A")),
             'B': len(os.listdir(directory+"/B")),
             'C': len(os.listdir(directory+"/C")),
             'D': len(os.listdir(directory+"/D")),
             'E': len(os.listdir(directory+"/E")),
             'F': len(os.listdir(directory+"/F")),
             'G': len(os.listdir(directory+"/G")),
             'H': len(os.listdir(directory+"/H")),
             'I': len(os.listdir(directory+"/I")),
             'J': len(os.listdir(directory+"/J")),
             'K': len(os.listdir(directory+"/K")),
             'L': len(os.listdir(directory+"/L")),
             'M': len(os.listdir(directory+"/M")),
             'N': len(os.listdir(directory+"/N")),
             'O': len(os.listdir(directory+"/O")),
             'P': len(os.listdir(directory+"/P")),
             'Q': len(os.listdir(directory+"/Q")),
             'R': len(os.listdir(directory+"/R")),
             'S': len(os.listdir(directory+"/S")),
             'T': len(os.listdir(directory+"/T")),
             'U': len(os.listdir(directory+"/U")),
             'V': len(os.listdir(directory+"/V")),
```

Figure 5 Getting a count of the images in the alphabet

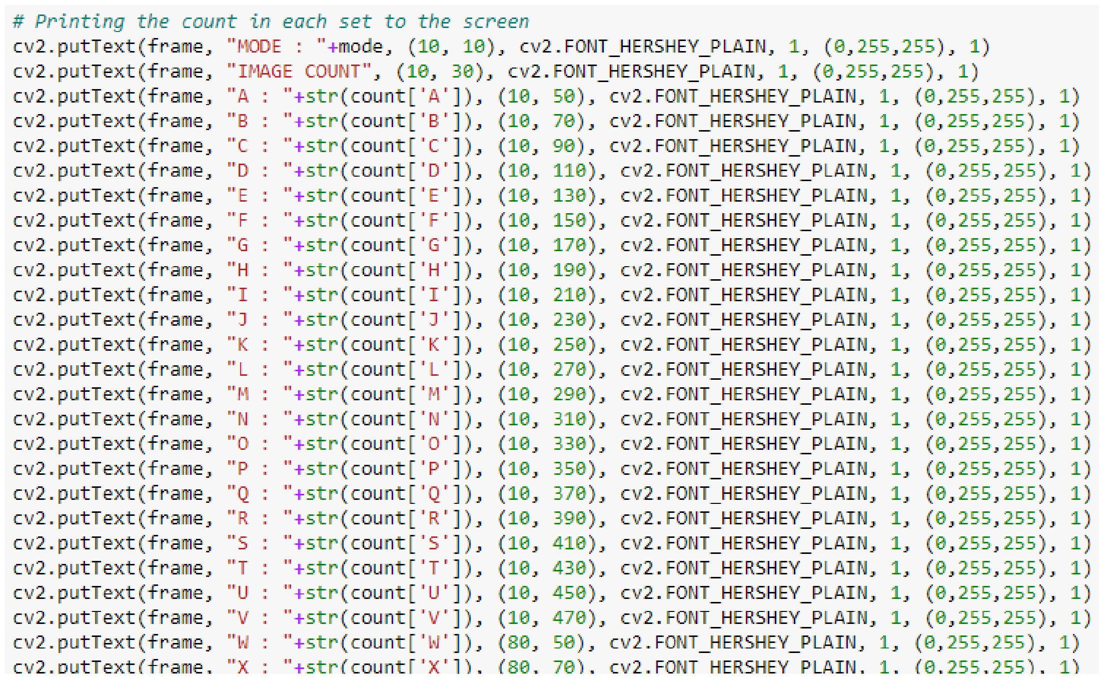


```python
cap = cv2.VideoCapture(0)
```

```python
while True:
    _, frame = cap.read()
    # Simulating mirror image
    frame = cv2.flip(frame, 1)

    # Getting count of existing images
    count = {'0': len(os.listdir(directory+"/0")),
             '1': len(os.listdir(directory+"/1")),
             '2': len(os.listdir(directory+"/2")),
             '3': len(os.listdir(directory+"/3")),
             '4': len(os.listdir(directory+"/4")),
             '5': len(os.listdir(directory+"/5")),
             '6': len(os.listdir(directory+"/6")),
             '7': len(os.listdir(directory+"/7")),
             '8': len(os.listdir(directory+"/8")),
             '9': len(os.listdir(directory+"/9")),
             '10': len(os.listdir(directory+"/10"))
```

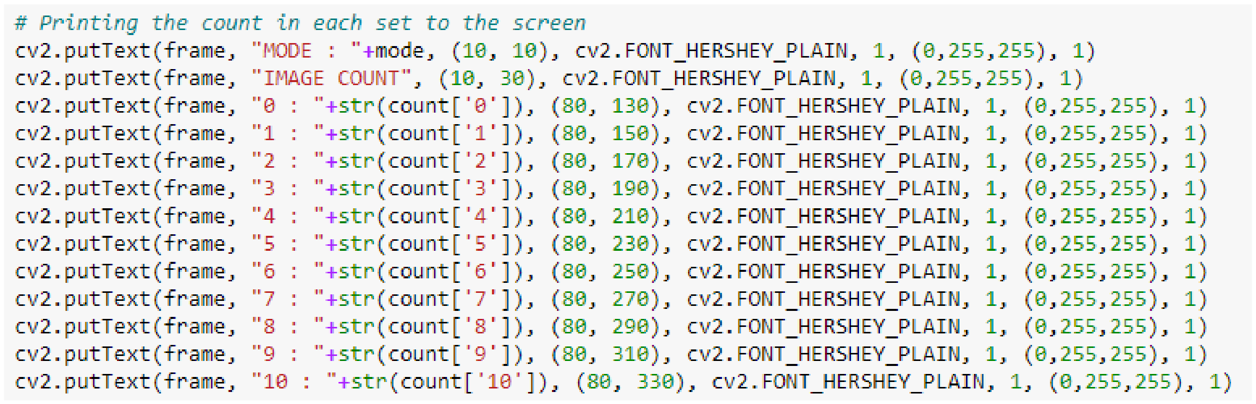
Figure 6 Getting a count of the images for Numbers

An image count was given upon capturing image for each class. The count was made to display on the screen while capturing an image for each gesture.



```python
# Printing the count in each set to the screen
cv2.putText(frame, "MODE : "+mode, (10, 10), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "IMAGE COUNT", (10, 30), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "A : "+str(count['A']), (10, 50), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "B : "+str(count['B']), (10, 70), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "C : "+str(count['C']), (10, 90), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "D : "+str(count['D']), (10, 110), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "E : "+str(count['E']), (10, 130), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "F : "+str(count['F']), (10, 150), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "G : "+str(count['G']), (10, 170), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "H : "+str(count['H']), (10, 190), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "I : "+str(count['I']), (10, 210), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "J : "+str(count['J']), (10, 230), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "K : "+str(count['K']), (10, 250), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "L : "+str(count['L']), (10, 270), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "M : "+str(count['M']), (10, 290), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "N : "+str(count['N']), (10, 310), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "O : "+str(count['O']), (10, 330), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "P : "+str(count['P']), (10, 350), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "Q : "+str(count['Q']), (10, 370), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "R : "+str(count['R']), (10, 390), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "S : "+str(count['S']), (10, 410), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "T : "+str(count['T']), (10, 430), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "U : "+str(count['U']), (10, 450), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "V : "+str(count['V']), (10, 470), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "W : "+str(count['W']), (80, 50), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "X : "+str(count['X']), (80, 70), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
```

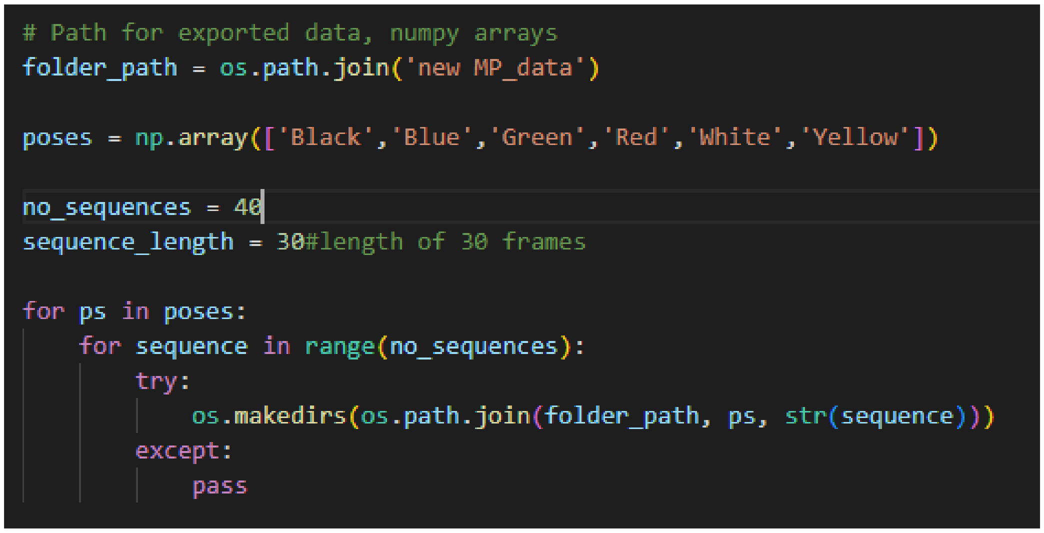
Figure 7 Display Image count on screen for data collection



```
# Printing the count in each set to the screen
cv2.putText(frame, "MODE : "+mode, (10, 10), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "IMAGE COUNT", (10, 30), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "0 : "+str(count['0']), (80, 130), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "1 : "+str(count['1']), (80, 150), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "2 : "+str(count['2']), (80, 170), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "3 : "+str(count['3']), (80, 190), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "4 : "+str(count['4']), (80, 210), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "5 : "+str(count['5']), (80, 230), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "6 : "+str(count['6']), (80, 250), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "7 : "+str(count['7']), (80, 270), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "8 : "+str(count['8']), (80, 290), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "9 : "+str(count['9']), (80, 310), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "10 : "+str(count['10']), (80, 330), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
```

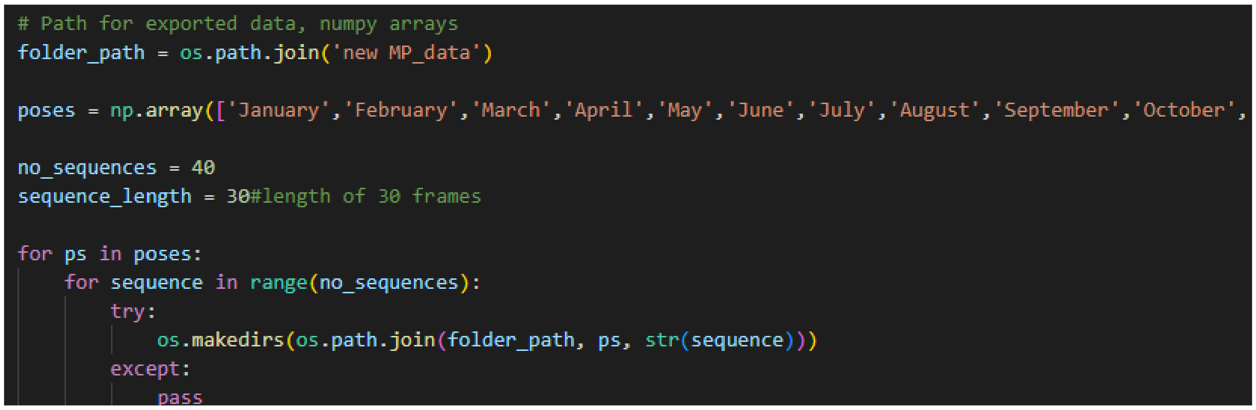
Figure 8 Display image count on screen for Numbers

Dynamic gestures were collected with a sequence of videos with 40 iterations and a sequence length of 30. The collected images were stored in the relevant directories in the form of a NumPy array. The dynamic gesture categories include: Colours, Days of the week, and the Months of the year



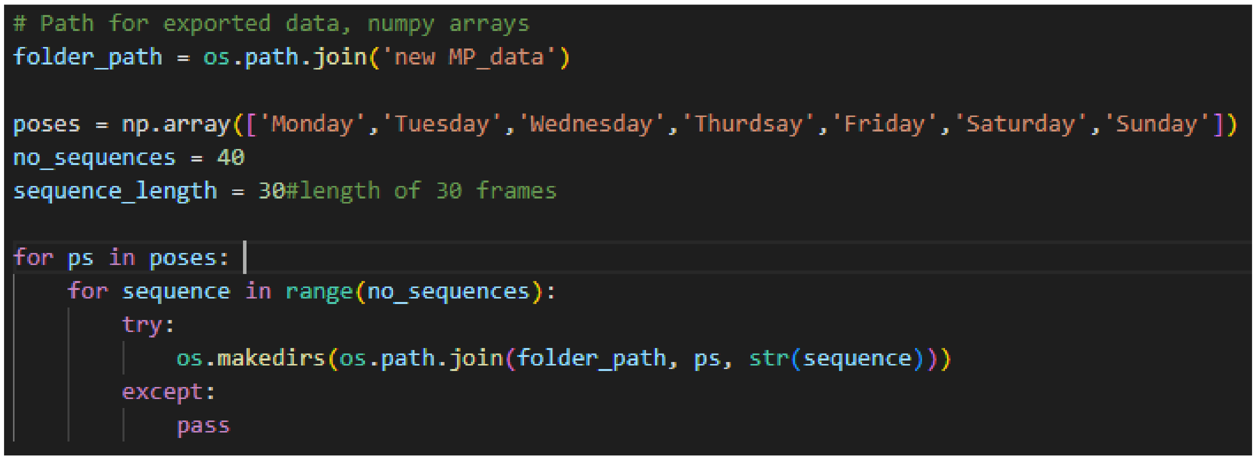
```
# Path for exported data, numpy arrays
folder_path = os.path.join('new MP_data')

poses = np.array(['Black','Blue','Green','Red','White','Yellow'])

no_sequences = 40
sequence_length = 30#length of 30 frames

for ps in poses:
    for sequence in range(no_sequences):
        try:
            os.makedirs(os.path.join(folder_path, ps, str(sequence)))
        except:
            pass
```

Figure 9 Collecting data for colours



```python
# Path for exported data, numpy arrays
folder_path = os.path.join('new MP_data')

poses = np.array(['January','February','March','April','May','June','July','August','September','October',

no_sequences = 40
sequence_length = 30#length of 30 frames

for ps in poses:
    for sequence in range(no_sequences):
        try:
            os.makedirs(os.path.join(folder_path, ps, str(sequence)))
        except:
            pass
```

Figure 10 Collecting data for months



```python
# Path for exported data, numpy arrays
folder_path = os.path.join('new MP_data')

poses = np.array(['Monday','Tuesday','Wednesday','Thurdsay','Friday','Saturday','Sunday'])
no_sequences = 40
sequence_length = 30#length of 30 frames

for ps in poses: 
    for sequence in range(no_sequences):
        try:
            os.makedirs(os.path.join(folder_path, ps, str(sequence)))
        except:
            pass
```

Figure 11 Collecting data for days of the week

For dynamic gesture detection, data was collected through a mediapipe holistic model which recognized the right hand, left hand, face and pose components of a figure and build landmarks on them

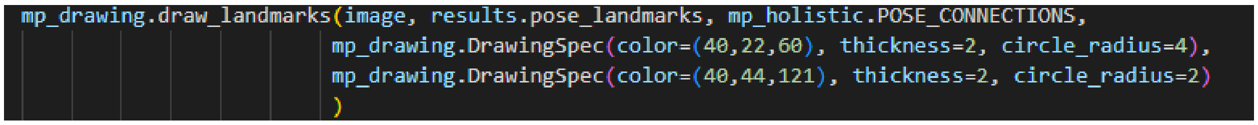

```
mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS,
                          mp_drawing.DrawingSpec(color=(40,22,60), thickness=2, circle_radius=4),
                          mp_drawing.DrawingSpec(color=(40,44,121), thickness=2, circle_radius=2)
                          )
```

Figure 14 Building landmarks for pose connections

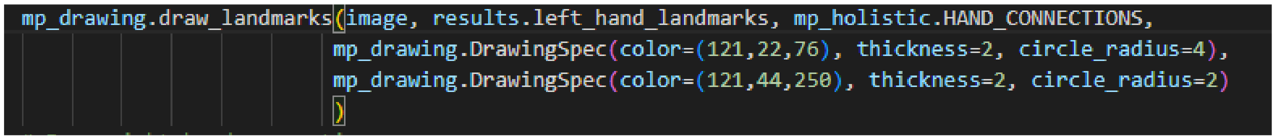

```
mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                          mp_drawing.DrawingSpec(color=(121,22,76), thickness=2, circle_radius=4),
                          mp_drawing.DrawingSpec(color=(121,44,250), thickness=2, circle_radius=2)
                          )
```

Figure 13 Building landmarks for left hand

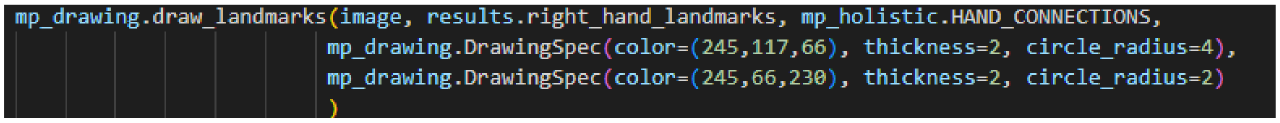
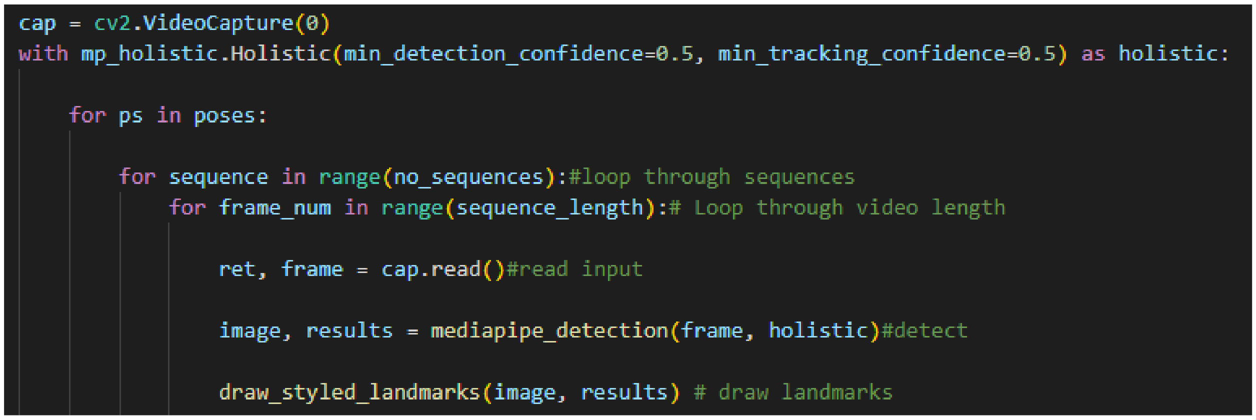

```
mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                          mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=4),
                          mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                          )
```

Figure 12 Building landmarks for right hand

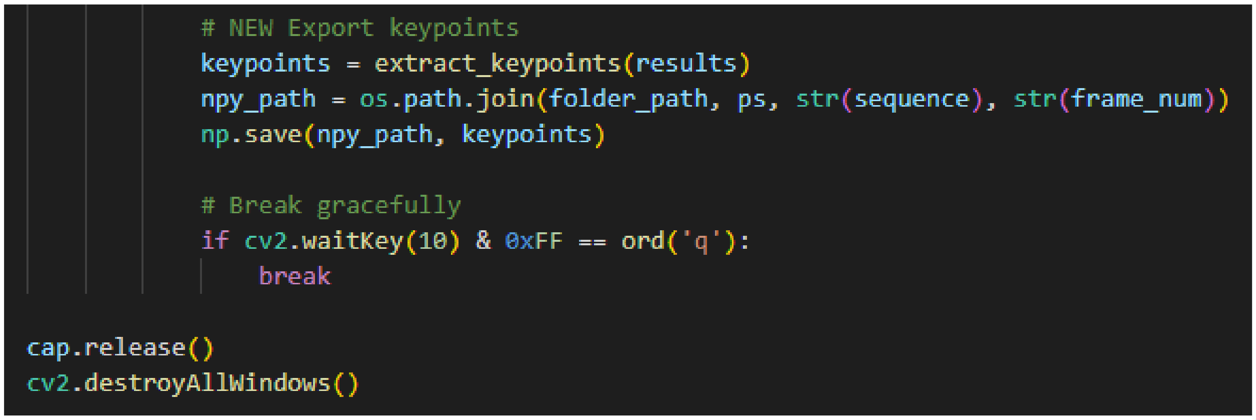
Video frames are captured through the OpenCV library function cv2.VideoCapture(). The code begins by initializing the holistic model. It’s made to loop through the predefined number of video sequences and the length of the video by allowing the user to give the input data.

The video frames will be captured for a specific period of time at each iteration while the user perform the gestures to the camera.



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

    for ps in poses:

        for sequence in range(no_sequences):#loop through sequences
            for frame_num in range(sequence_length):# Loop through video length

                ret, frame = cap.read()#read input

                image, results = mediapipe_detection(frame, holistic)#detect

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

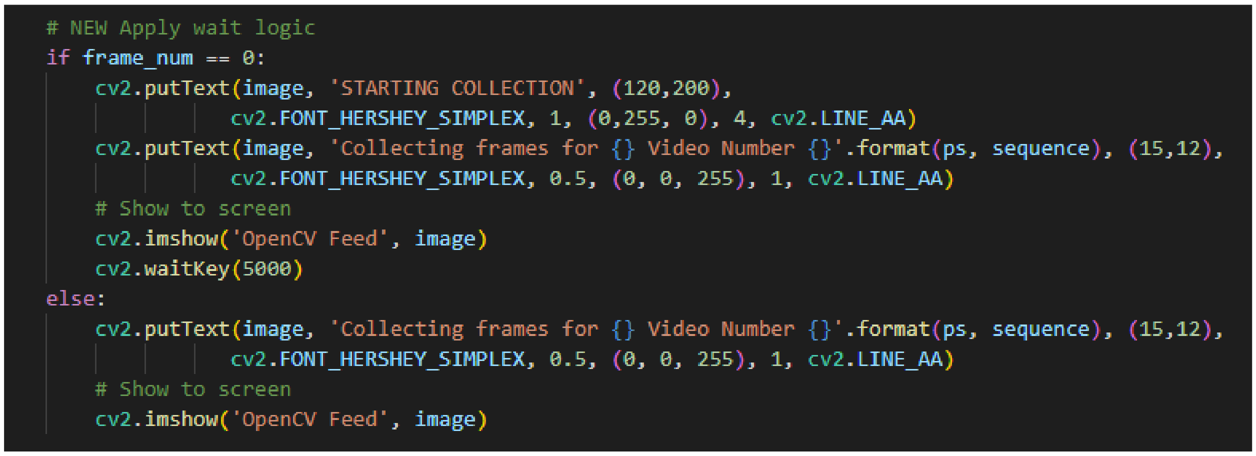
Figure 15 Collecting data through web camera



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

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

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

Figure 16 Defining the keypoints

A wait logic is applied after each iteration of the video. A countdown on the number of iterations and the name of the class is displayed on the top.



```python
# NEW Apply wait logic
if frame_num == 0:
    cv2.putText(image, 'STARTING COLLECTION', (120,200),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255, 0), 4, cv2.LINE_AA)
    cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(ps, sequence), (15,12),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
    # Show to screen
    cv2.imshow('OpenCV Feed', image)
    cv2.waitKey(5000)
else:
    cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(ps, sequence), (15,12),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
    # Show to screen
    cv2.imshow('OpenCV Feed', image)
```

Figure 17 Applying a wait logic after each iteration

3.2.2 Importing Libraries and packages

A few essential libraries were imported to support the data collection and training process. Libraries such as OpenCV and NumPy were mainly used due to the fact that the data was collected using the web camera.

- OpenCV
  Perform real-time computer vision techniques

- NumPy
  Perform mathematical operations on various array sizes

- OS Module
  Identify the relevant directories and fetch data from them.

- Mediapipe Holistic
  Pipeline which enables to track hand, face and pose components simultaneously by creating landmarks on them.

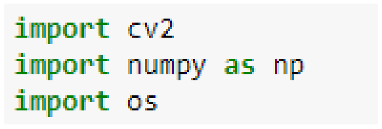

```
import cv2
import numpy as np
import os
```

Figure 18 Imported libraries for static

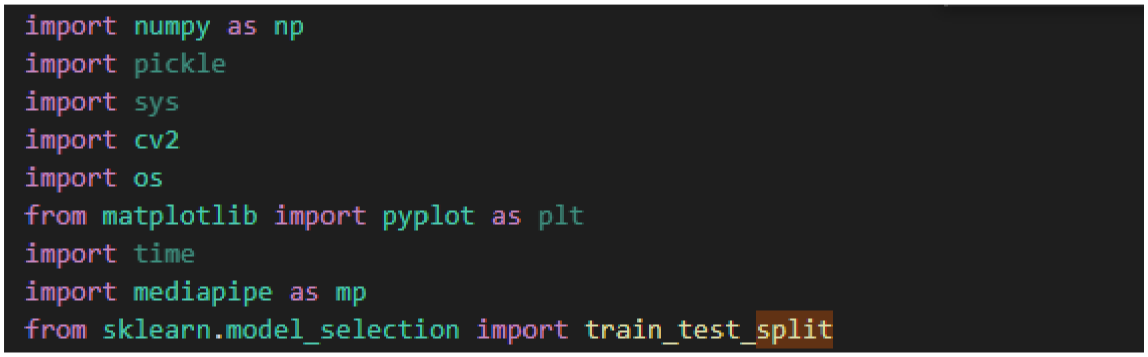

```
import numpy as np
import pickle
import sys
import cv2
import os
from matplotlib import pyplot as plt
import time
import mediapipe as mp
from sklearn.model_selection import train_test_split
```

Figure 19 Imported libraries for Dynamic gestures

Keras libraries and packages were imported in the model building process.

Keras models:

- Sequential model
  To build a sequential neural network. It was imported for both dynamic and static gesture detection.

Keras Layers:

- LSTM Layer
- Dense Layer
- Convolutional 2D
- Maxpooling 2D
- Flatten

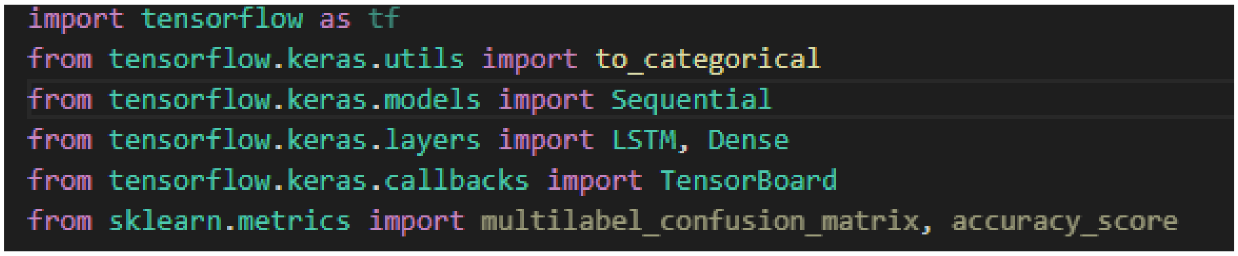
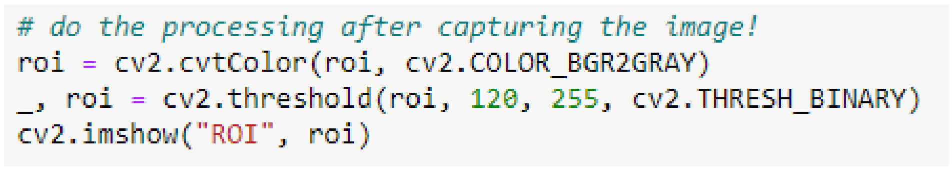

```
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import TensorBoard
from sklearn.metrics import multilabel_confusion_matrix, accuracy_score
```

Figure 20 Imported libraries and packages in Keras

3.2.3 Data Preprocessing

Through data preprocessing the collected images underwent a series of modifications to make the detections more accurate and efficient. Following data preprocessing techniques were performed:

- Colour scale conversion
  A color scale conversion was performed on the collected images by converting the RGB image to a greyscale format. By carrying out this process the color composition was reduced to two colors (Black and white).

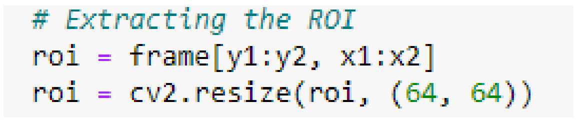


```
# do the processing after capturing the image!
roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
_, roi = cv2.threshold(roi, 120, 255, cv2.THRESH_BINARY)
cv2.imshow("ROI", roi)
```

Figure 21 Colour scale conversion

- Resizing

  It's essential to give a fixed shape to the images. As a result, image resizing is performed on each image The collected images were resized into a 64*64 pixel scale.

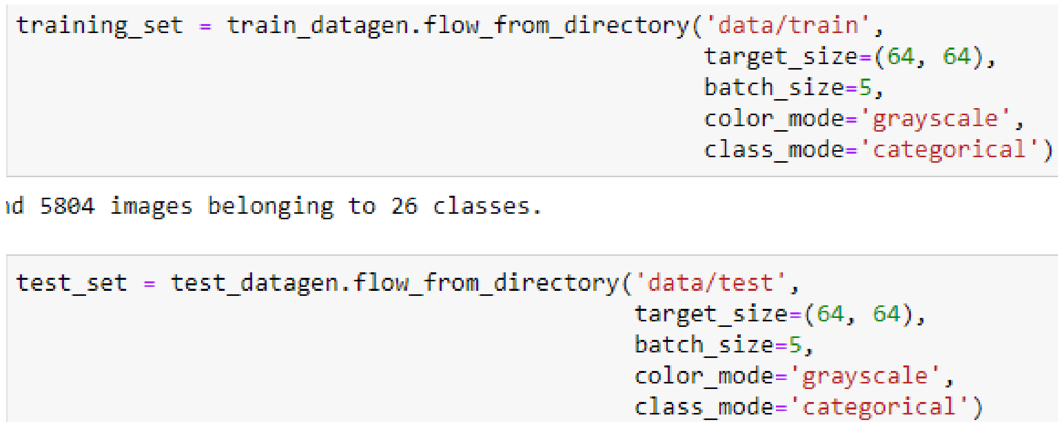


```
# Extracting the ROI
roi = frame[y1:y2, x1:x2]
roi = cv2.resize(roi, (64, 64))
```

Figure 22 Resizing images

3.2.4 Splitting the dataset

Considering the static gestures, the data was split into training and testing sets. 80% of the collected data was sent for training while the rest of the images were used for testing purposes.



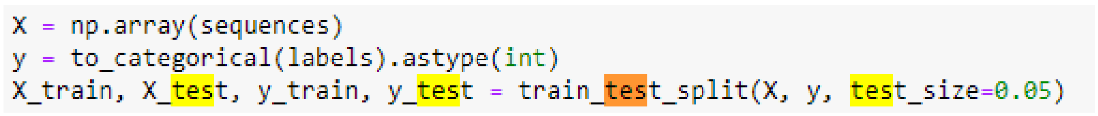
```
training_set = train_datagen.flow_from_directory('data/train',
                                                 target_size=(64, 64),
                                                 batch_size=5,
                                                 color_mode='grayscale',
                                                 class_mode='categorical')
```

```
d 5804 images belonging to 26 classes.
```

```
test_set = test_datagen.flow_from_directory('data/test',
                                            target_size=(64, 64),
                                            batch_size=5,
                                            color_mode='grayscale',
                                            class_mode='categorical')
```

Figure 23 Splitting the data into testing and training sets in CNN model

However, the data that was collected for the dynamic gestures through video sequences defined a test size of 0.05 containing 5% of validation data and 95% of training data.



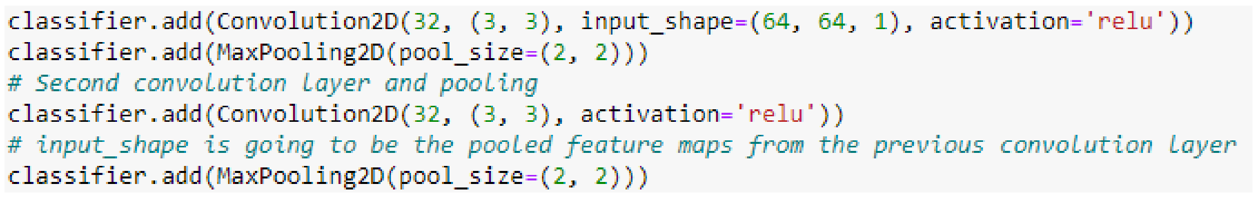
```
X = np.array(sequences)
y = to_categorical(labels).astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)
```

Figure 24 Figure 23 Splitting the data into testing and training sets in LSTM model

3.2.5 Model Building

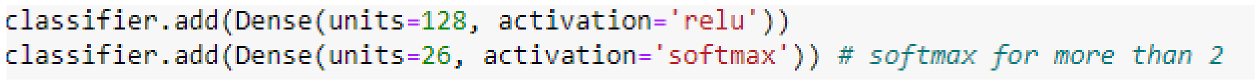
Model building is a crucial part of the implementation. The overall results depend on the appropriateness of the model that is chosen. Therefore, the overall accuracy could be further enhanced upon choosing a suitable model. Hence by doing several tests the best model that was chosen was able to provide adequate accuracy for each dynamic and static gesture categories.

Since the Static gestures focus on image classification the model is trained using Tensorflow and Keras with the Convolutional Neural Network. The Convolutional Neural Network includes 2 sets of Convolutional and Pooling layers and 2 Dense Layers.



```
classifier.add(Convolution2D(32, (3, 3), input_shape=(64, 64, 1), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))
# Second convolution layer and pooling
classifier.add(Convolution2D(32, (3, 3), activation='relu'))
# input_shape is going to be the pooled feature maps from the previous convolution layer
classifier.add(MaxPooling2D(pool_size=(2, 2)))
```

Figure 25 Adding a Convolutional and Pooling Layer



```
classifier.add(Dense(units=128, activation='relu'))
classifier.add(Dense(units=26, activation='softmax')) # softmax for more than 2
```

Figure 26 Adding a fully connected layer

These layers consist of 32 filters. The input image of the hand gesture is passed through the Convolutional and Pooling layers to extract specific features out of the image. Since the output of these layers is in the form of a 2-D array it will be converted into a linear vector using the Flatten () function and is passed to the Dense layers.

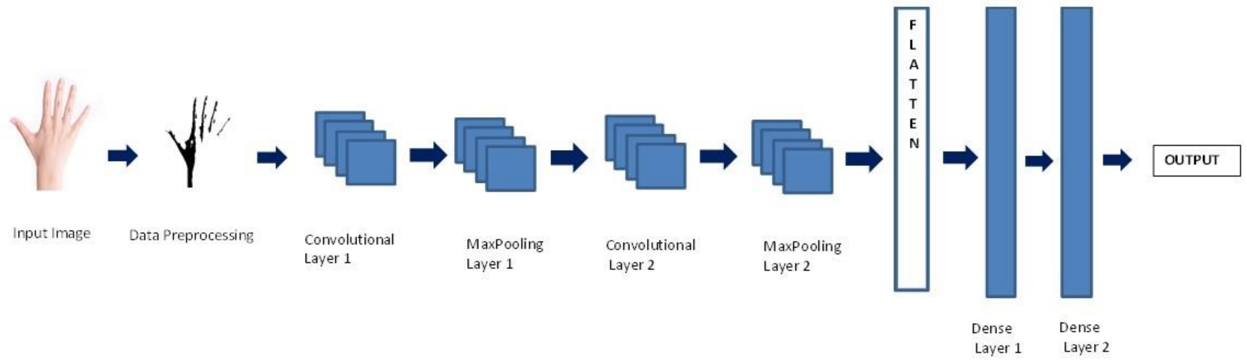
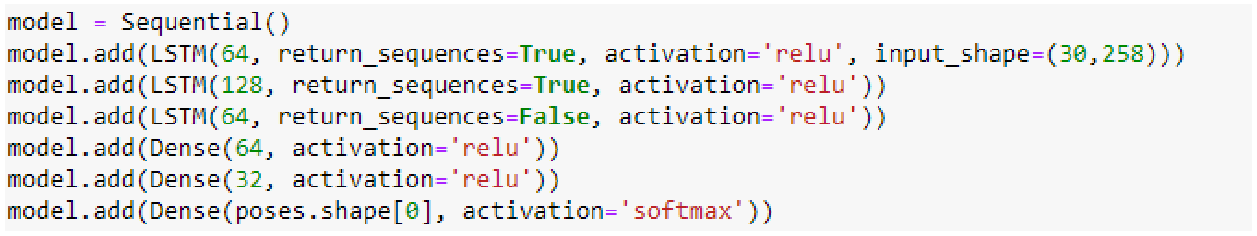


Figure 27 CNN Architecture Diagram

Dynamic signs, on the other hand, are trained with the aid of an LSTM model. The model consists of 3 LSTM layers and 3 Dense layers. Videos are captured through the web camera and are collected in 40 iterations per gesture. The collected images were trained with roughly around 200 epochs which vary depending on the number of classes for each category

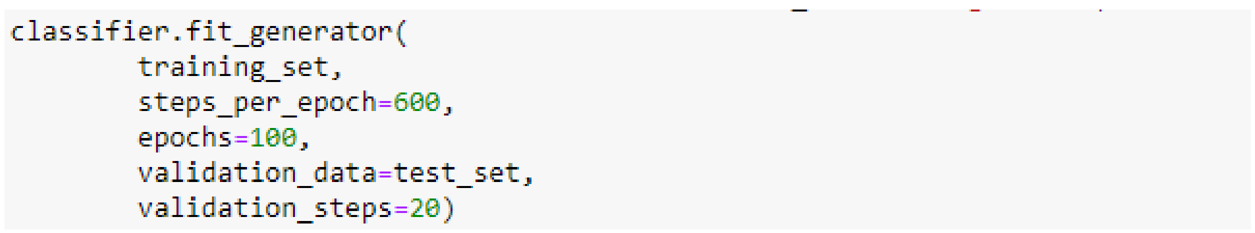


```
model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(30,258)))
model.add(LSTM(128, return_sequences=True, activation='relu'))
model.add(LSTM(64, return_sequences=False, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(poses.shape[0], activation='softmax'))
```

Figure 28 Adding LSTM and Dense Layers

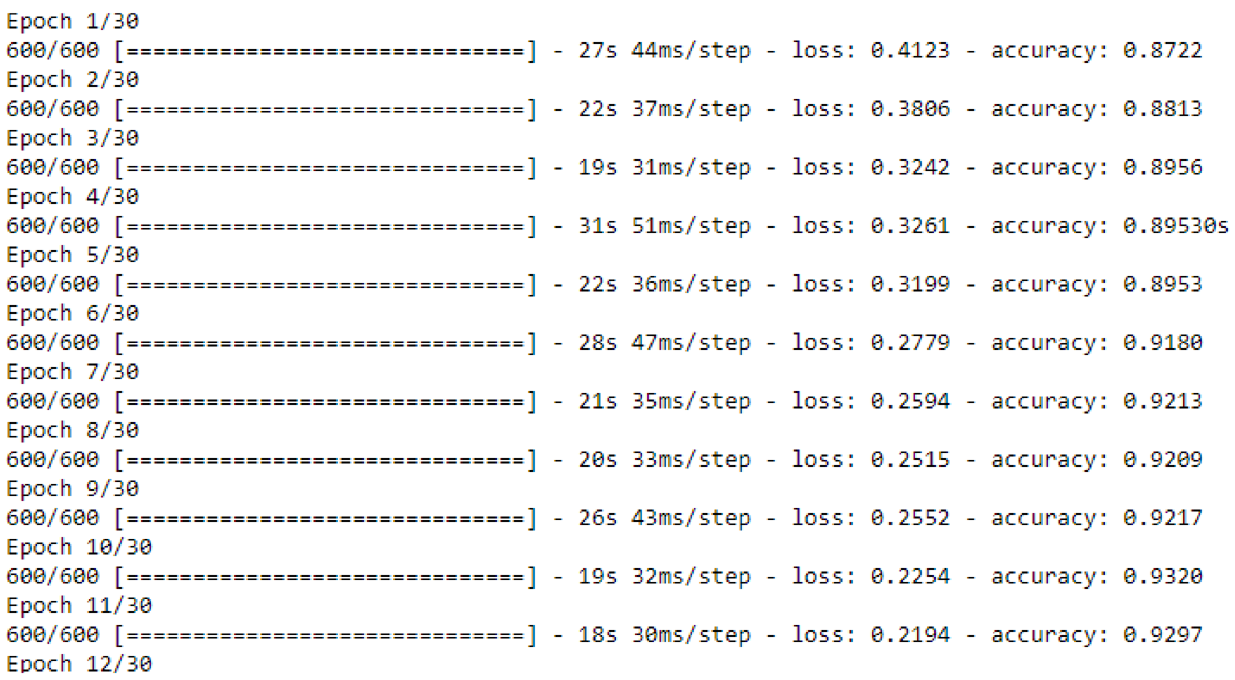
3.2.6 Training the data

Each class in the static gesture category were trained with 100 epochs, containing 600 steps per epoch. Dynamic gestures were trained with 300 epochs. The number of epochs were slightly more or less for each category depending on the number of classes it contained. Training of video sequences required to train with a higher number of epochs compared to image frames in order to obtain a higher accuracy.



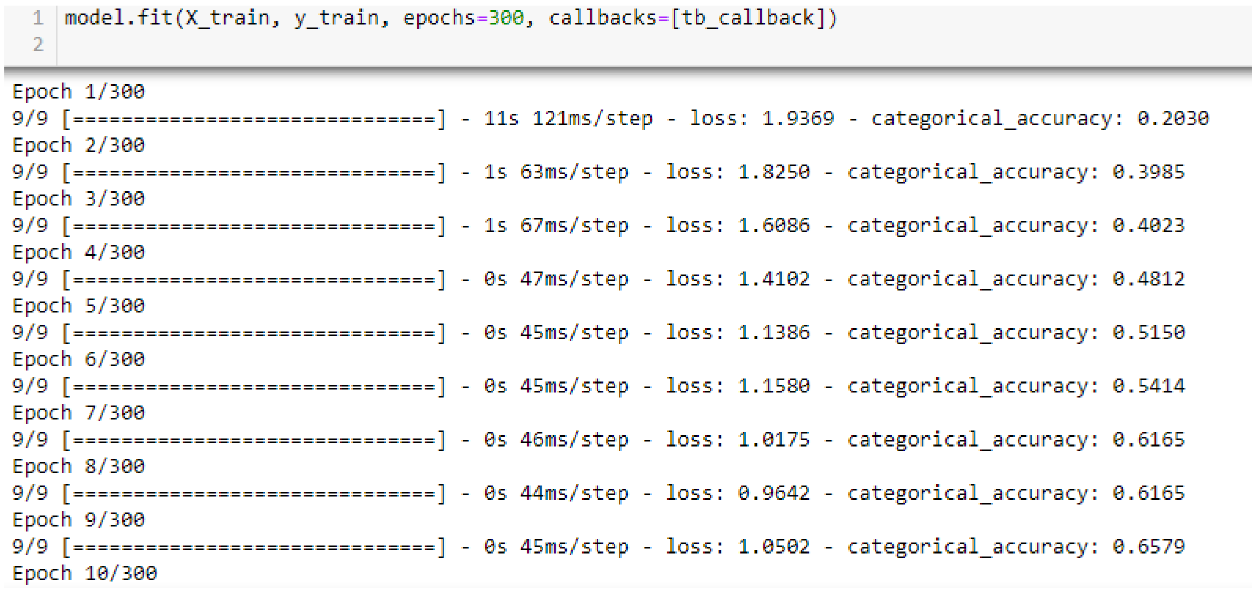
```
classifier.fit_generator(
        training_set,
        steps_per_epoch=600,
        epochs=100,
        validation_data=test_set,
        validation_steps=20)
```

Figure 29 Training CNN model



```
Epoch 1/30
600/600 [==============================] - 27s 44ms/step - loss: 0.4123 - accuracy: 0.8722
Epoch 2/30
600/600 [==============================] - 22s 37ms/step - loss: 0.3806 - accuracy: 0.8813
Epoch 3/30
600/600 [==============================] - 19s 31ms/step - loss: 0.3242 - accuracy: 0.8956
Epoch 4/30
600/600 [==============================] - 31s 51ms/step - loss: 0.3261 - accuracy: 0.89530s
Epoch 5/30
600/600 [==============================] - 22s 36ms/step - loss: 0.3199 - accuracy: 0.8953
Epoch 6/30
600/600 [==============================] - 28s 47ms/step - loss: 0.2779 - accuracy: 0.9180
Epoch 7/30
600/600 [==============================] - 21s 35ms/step - loss: 0.2594 - accuracy: 0.9213
Epoch 8/30
600/600 [==============================] - 20s 33ms/step - loss: 0.2515 - accuracy: 0.9209
Epoch 9/30
600/600 [==============================] - 26s 43ms/step - loss: 0.2552 - accuracy: 0.9217
Epoch 10/30
600/600 [==============================] - 19s 32ms/step - loss: 0.2254 - accuracy: 0.9320
Epoch 11/30
600/600 [==============================] - 18s 30ms/step - loss: 0.2194 - accuracy: 0.9297
Epoch 12/30
```

Figure 30 Running epochs for the CNN model



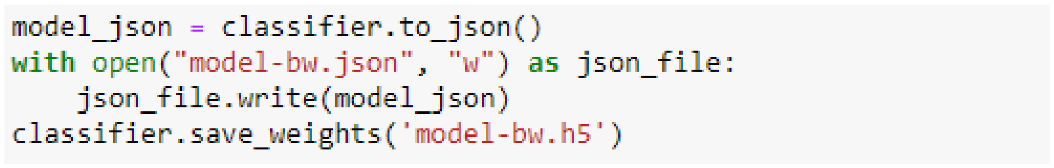
```
model.fit(X_train, y_train, epochs=300, callbacks=[tb_callback])
```

```
Epoch 1/300
9/9 [==============================] - 11s 121ms/step - loss: 1.9369 - categorical_accuracy: 0.2030
Epoch 2/300
9/9 [==============================] - 1s 63ms/step - loss: 1.8250 - categorical_accuracy: 0.3985
Epoch 3/300
9/9 [==============================] - 1s 67ms/step - loss: 1.6086 - categorical_accuracy: 0.4023
Epoch 4/300
9/9 [==============================] - 0s 47ms/step - loss: 1.4102 - categorical_accuracy: 0.4812
Epoch 5/300
9/9 [==============================] - 0s 45ms/step - loss: 1.1386 - categorical_accuracy: 0.5150
Epoch 6/300
9/9 [==============================] - 0s 45ms/step - loss: 1.1580 - categorical_accuracy: 0.5414
Epoch 7/300
9/9 [==============================] - 0s 46ms/step - loss: 1.0175 - categorical_accuracy: 0.6165
Epoch 8/300
9/9 [==============================] - 0s 44ms/step - loss: 0.9642 - categorical_accuracy: 0.6165
Epoch 9/300
9/9 [==============================] - 0s 45ms/step - loss: 1.0502 - categorical_accuracy: 0.6579
Epoch 10/300
```

Figure 31 Running epochs for the LSTM model

3.2.7 Saving weights

In general, it takes a considerable amount of time for all the epochs to run and end the training process of a model. If the weights of the model are not saved anywhere, we may have to re-train the model each time. Therefore, it’s optimum to reuse the trained model, after obtaining adequate weights after a long training process. As a result, each time the model goes through a training process it’s weights is saved in an HDF5 format. The model will get saved in a directory and we can simply load it for detection



```
model_json = classifier.to_json()
with open("model-bw.json", "w") as json_file:
    json_file.write(model_json)
classifier.save_weights('model-bw.h5')
```

Figure 32 Saving weights of the trained model

3.2.8 Avatar Model Building

The gesture movements for each sign was operated using prebuilt avatar models that were provided by ‘DeepMotion’ and ‘3D Hand Draw’ applications.

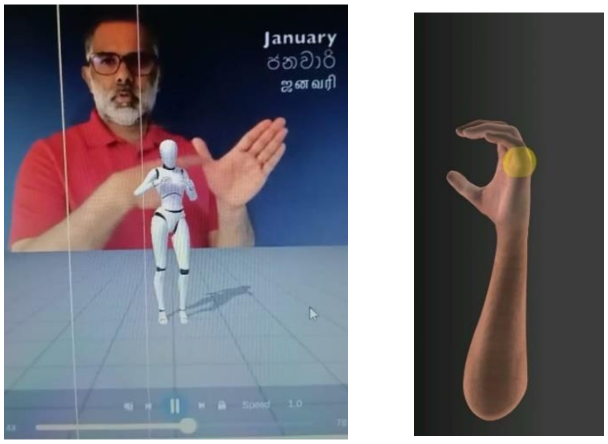
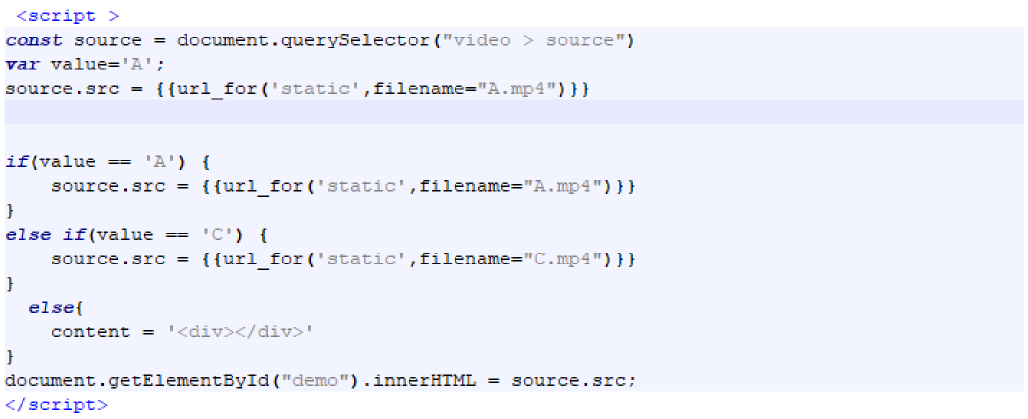


Figure 33 Trained avatar models

Upon obtaining traced video clips of the avatar models, they are being displayed to the user depending on the input hand gesture that was detected. The video clip will be mapped with the relevant sign and is displayed to the user. The implementation for this part was done using HTML and JavaScript. A Flask API was used to pass the output results.



```
 <script >
const source = document.querySelector("video > source")
var value='A';
source.src = {{url_for('static',filename="A.mp4")}}


if(value == 'A') {
    source.src = {{url_for('static',filename="A.mp4")}}
}
else if(value == 'C') {
    source.src = {{url_for('static',filename="C.mp4")}}
}
  else{
    content = '<div></div>'
}
document.getElementById("demo").innerHTML = source.src;
</script>
```

Figure 34 Retrieve video and displaying in frontend

3.3 Testing

In the testing phase, a word from each category was tested to verify that the detection happens accurately. Upon detection, its tested to ensure that the correct video and the text is displayed in the UI.

Table 2 Test Case 1

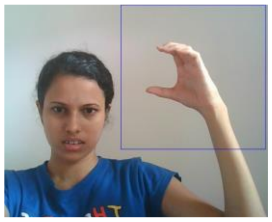
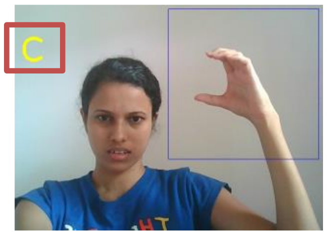
TEST CASE 01	
Category	The Alphabet
Description	Testing the conversion of a given letter in sign language to text
Input	
Expected Output	Detect the hand gesture and display the name as ‘C’
Actual Output	
Test Status	Pass

Table 3: Test case 2

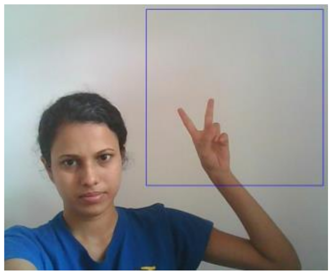
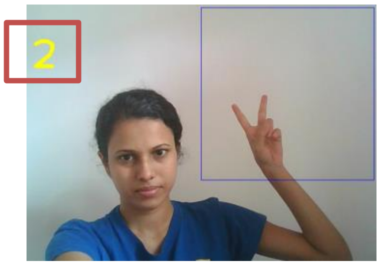
TEST CASE 02	
Category	Numbers[0 -10]
Description	Testing a given number from sign language to text conversion
Input	
Expected Output	Detect the hand gesture and display the number as ‘2’
Actual Output	
Test Status	Pass

Table 4: Test case 3

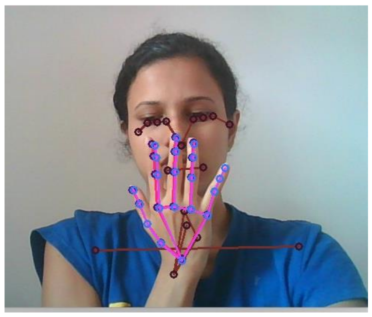
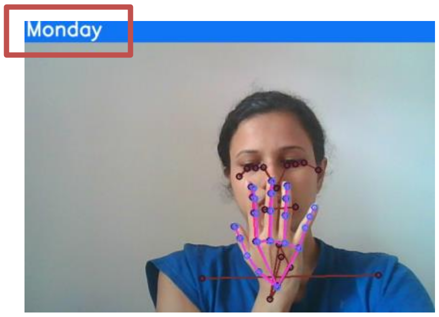
TEST CASE 03	
Category	Days of the Week
Description	Testing sign language to text conversion, given a day in the week
Input	
Expected Output	Detect the hand gesture and display the name
Actual Output	 Monday
Test Status	Pass

Table 5: Test case 4

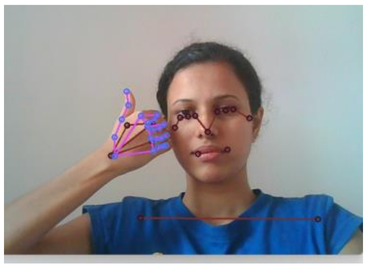
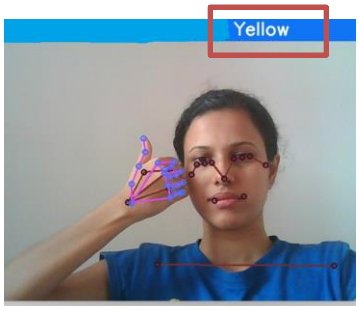
TEST CASE 04	
Category	Colours
Description	Testing sign language to text conversion, given a gesture of a colour
Input	
Expected Output	Detect the hand gesture and display the name as ‘Yellow’
Actual Output	
Test Status	Pass

Table 6: Test case 5

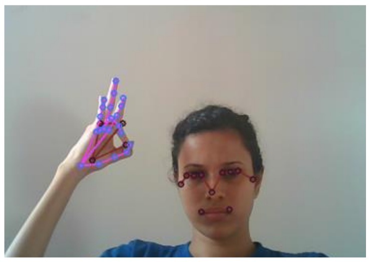
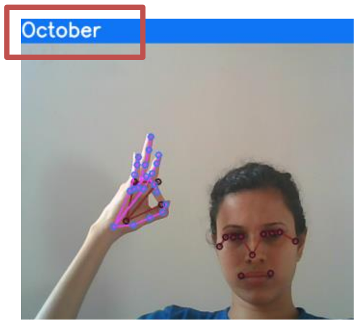
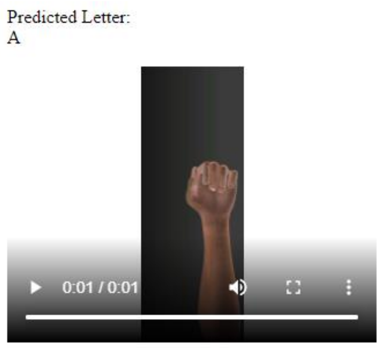
TEST CASE 05	
Category	Months of the Year
Description	Testing sign language to text conversion
Input	
Expected Output	Detect the hand gesture and display the name of the months as 'October'
Actual Output	 October
Test Status	Pass

Table 7: Test case 6

TEST CASE 06	
Description	Retrieve the relevant video according to the given gesture
Input	Giving the hand gesture for the letter 'A'
Expected Output	Relevant video clip denoting the avatar for letter 'A'
Actual Output	 Predicted Letter: A
Test Status	Pass

4 RESULTS AND DISCUSSION

4.1 Results

In the categories that were chosen to train, the Alphabet, and the numbers [1-10] contain static gestures while the Months and Days include dynamic gestures altogether. Based on the category a suitable model was chosen and was able to obtain a good accuracy for each component. The following diagram displays the accuracies and the model that was used for each category of signs.

Table 8: Accuracy comparison for each category of words

Category	Type	Model	Accuracy
Alphabet	Static	CNN	84%
Numbers	Static	CNN	70.82%
Months	Dynamic	LSTM	79.82%
Days	Dynamic	LSTM	100%

4.2 Research Findings

One of the most compelling findings of this research was evaluating the best model and obtaining a good accuracy. Since the research was carried out using both image and action detection categories (static and dynamic), it was essential to find out the most suitable model which gives the best accuracy for both categories.

- Evaluating the best model

The static signs were trained using video processing through an LSTM model initially. However, after testing with a CNN model it was later verified that the static signs could be trained using image processing with a less number of training data and a higher accuracy. Therefore, the signs in the Alphabet which were initially trained with an LSTM model were later trained and tested using a CNN model since this model is more beneficial to obtain a good accuracy for static images. As a result, the numbers from 1-10 which also includes static gestures were trained by the CNN model followed by the previous conclusion. The figure below is a comparison of the models with the accuracies.

Table 9: Comparison between CNN and LSTM models for dynamic gestures

Category	Model	Accuracy
Alphabet	LSTM	51.7%
	CNN	84%

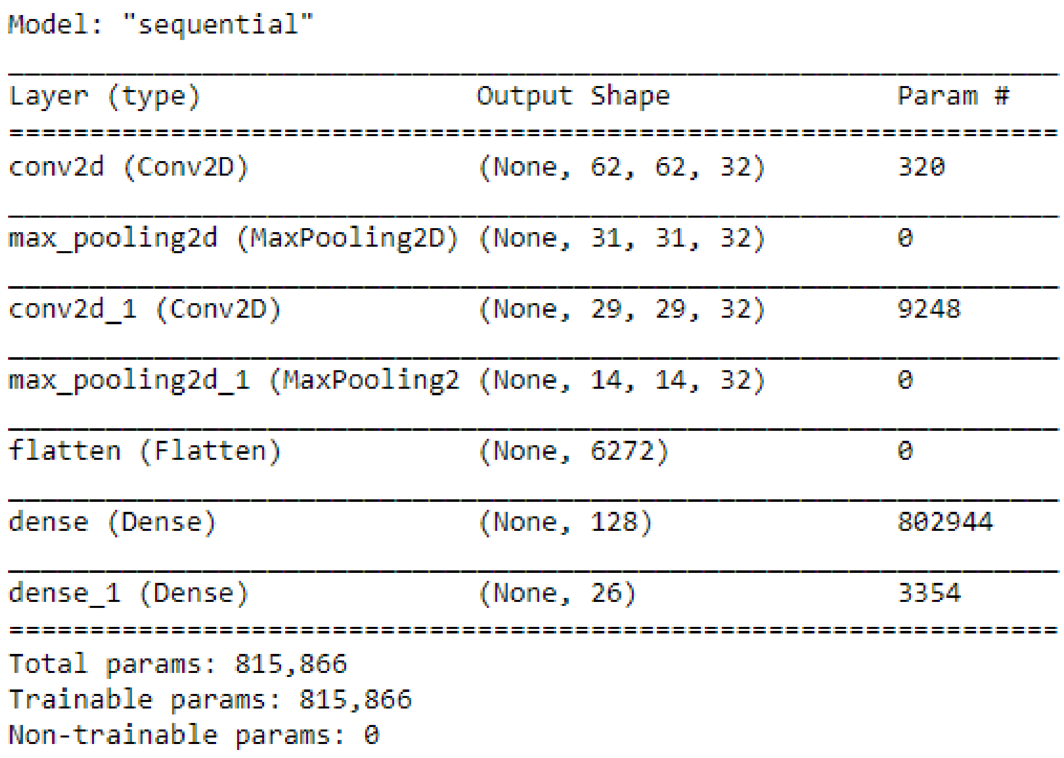
This however concludes that; image processing could be done using a CNN model with a better accuracy compared to a LSTM model.

- Obtaining a good accuracy

  While training the models the number of epochs were initially set to 50, however when the number of epochs were increased to a higher number the accuracy was increased. As a result, it was found out that increasing the number of epochs can be used to used to obtain an adequate accuracy.

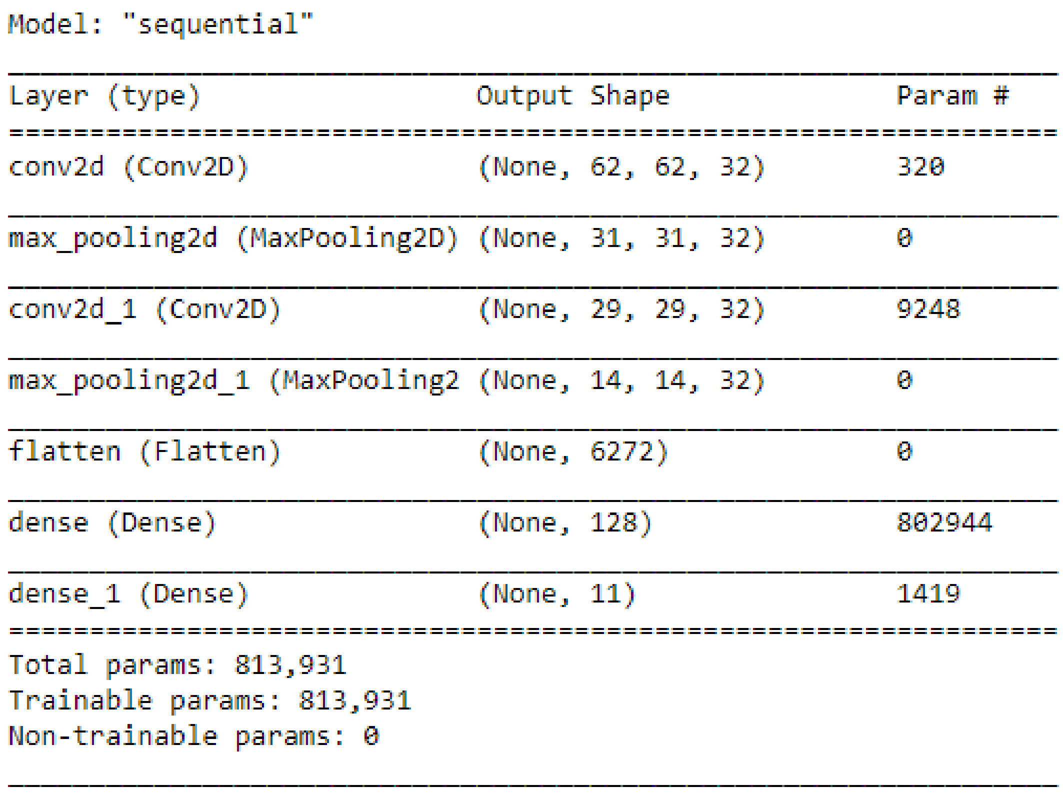
4.3 Discussion

- Alphabet
  Trained using a CNN model using 2 sets of convolutional layers, 2 sets of Maxpooling layers, and 2 dense layers. The figure below shows a summary of the model including a count of trainable and non-trainable params for each set.



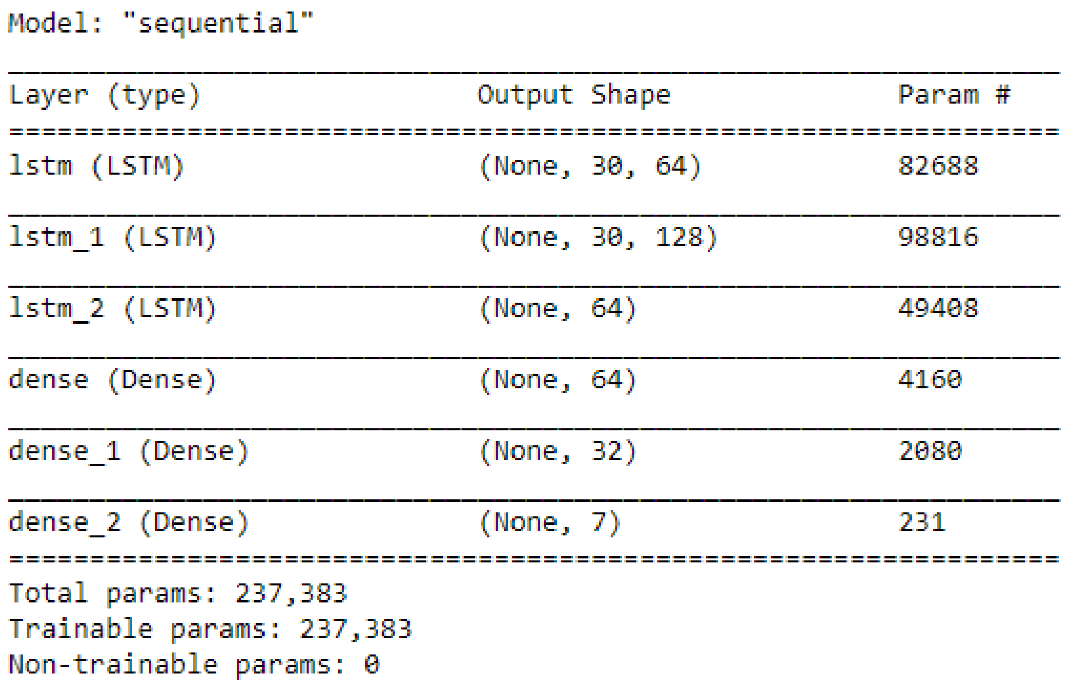
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 62, 62, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 31, 31, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 29, 29, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 128)               802944
_________________________________________________________________
dense_1 (Dense)              (None, 26)                3354
=================================================================
Total params: 815,866
Trainable params: 815,866
Non-trainable params: 0
```

- Numbers [0-10]
  Numbers belonging to the static sign category were also trained using the same CNN model as the Alphabet. It includes the same number of layers as mentioned above. However, the total parameters are slightly reduced compared to the previous model



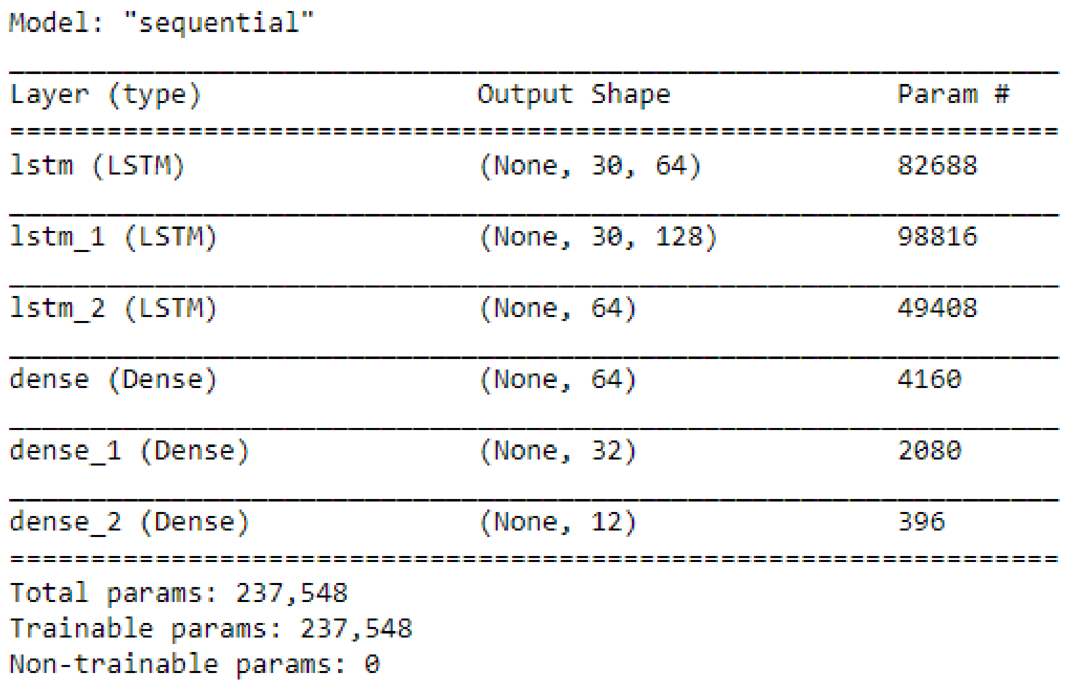
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 62, 62, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 31, 31, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 29, 29, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 128)               802944
_________________________________________________________________
dense_1 (Dense)              (None, 11)                1419
=================================================================
Total params: 813,931
Trainable params: 813,931
Non-trainable params: 0
_________________________________________________________________
```

- Days
  Trained using 3 sets of LSTM layers and 3 sets of Dense layers for action detection. The figure below is a summary of the trained model for the 'Days' category.



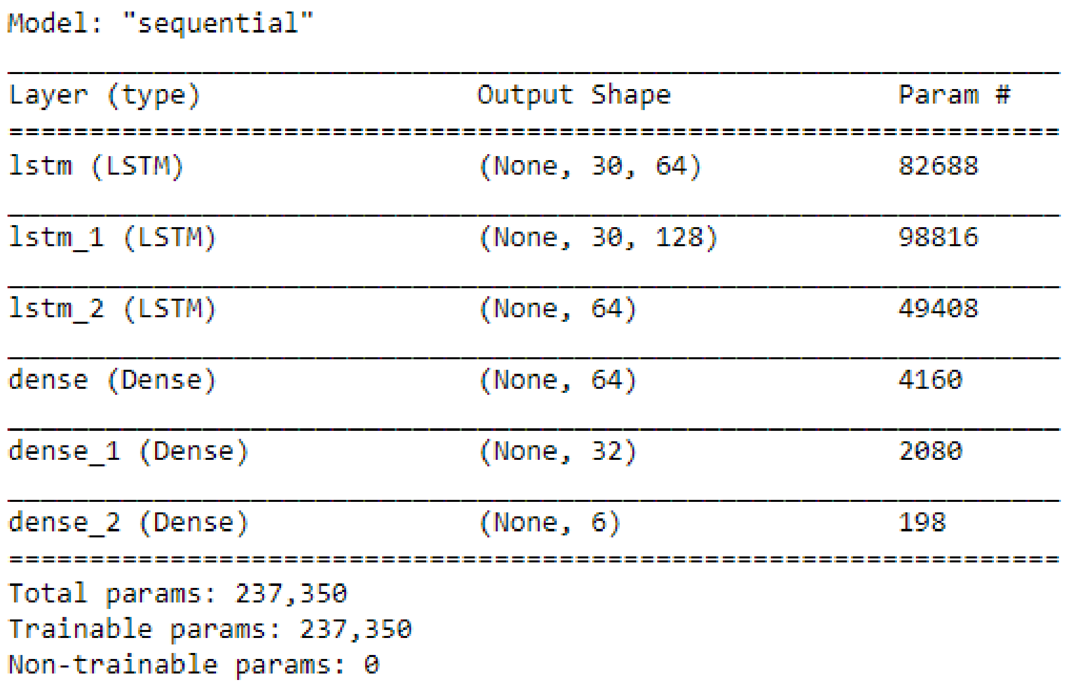
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 30, 64)            82688
_________________________________________________________________
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, 7)                 231
=================================================================
Total params: 237,383
Trainable params: 237,383
Non-trainable params: 0
```

- Months
  Months of the year contains another set of dynamic gestures. It was also trained using 3 sets of LSTM layers and 3 sets of Dense layers. The parameter count for this model is slightly greater than the model trained for ‘Days’



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 30, 64)            82688
_________________________________________________________________
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, 12)                396
=================================================================
Total params: 237,548
Trainable params: 237,548
Non-trainable params: 0
```

- Colours



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 30, 64)            82688
_________________________________________________________________
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, 6)                 198
=================================================================
Total params: 237,350
Trainable params: 237,350
Non-trainable params: 0
```

5 CONCLUSION

Bringing out a system that can translate SSL into ASL can be beneficial for a vast majority of the HIC to widen their knowledge on a non-native language. Such systems could also broaden the opportunity to learn ASL disregarding the difficulties of finding resources or interpreters.

Most of the research that has been carried out previously translates sign language into a spoken language by translating it to a text format. However, this research is further expanded by translating the converted text into a different sign language followed by a pretrained avatar model which denote the gestures to the user. Certain image and video processing techniques are used to detect the signs that are input to the system for detection. The models are selected after training with several comparisons to obtain an adequate accuracy. This research could be further expanded into translating higher number of words and could be converted into various other languages.

6 REFERENCES

[1] "World Health Organization," April 2021. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss.

[2] "National Association Of The Deaf," [Online]. Available: https://www.nad.org/resources/american-sign-language/what-is-american-sign-language.

[3] "a.i media," [Online]. Available: ((https://www.ai-media.tv/ai-media-blog/sign-language-alphabets-from-around-the-world/)However.

[4] K. Pandey, "South Asia accounts for over one-fourth of the world's hearing impaired: WHO," *DownToEarth*, March 2014.

[5] A. Shukla, M. Harper, E. Pederson, A. Gorman, J. J. Suen, C. Price, J. Applebaum, M. Hoyer and N. Reed, "Hearing Loss, Loneliness, and Social Isolation: A Systematic Review," *Sage Journals*, vol. 162, no. 5, pp. 622-633, March 2020.

[6] P. Fernando and P. Wimalaratne, "Sign Language Translation Approach to Sinhalese Language," 2016.

[7] "Report on supply and demand for Auslan interpreters," 2004.

[8] V. D. Andrade, "Deafness carries a huge cost burden: economic as well as personal," March 2017. [Online]. Available: https://theconversation.com/deafness-carries-a-huge-cost-burden-economic-as-well-as-personal-73532.

[9] M. Purse, "Deaf Community and Mental Health Care," June 2021. [Online]. Available: https://www.verywellmind.com/mental-health-issues-in-the-deaf-community-380577.

[10] Y. Perera, N. Jayalath, S. Tissera, O. Bandara and S. Thelijjagoda, "Intelligent mobile assistant for hearing impairers to interact with the society in Sinhala language," 2017.

[11] D. M. Kumar, K. Bavanraj, S. Thavanandan, G. Bastiansz, S. Harshanath and J. Alosious, "EasyTalk: A Translator for Sri Lankan Sign Language using Machine Learning and Artificial Intelligence," 2020.

[12] I. Dissanayaka, P. Wickramanayake, M. Mudunkotuwa and P. Fernando, ""Utalk: Sri Lankan Sign Language Converter Mobile App using Image Processing and Machine".

[13] "Static Sign Language Recognition Using Deep Learning".

[14] ""Development of a CG-animation system for sign language communication between different languages".

[15] "A Machine Translation System from English to American Sign Language".

[16] "Semantic analyses of text to translate to Russian sign language".

7 APPENDICES

Survey link:

https://docs.google.com/forms/d/e/1FAIpQLSd3GQeVu3_k8lL2uvtTy2hGc8-he1msLWR_5Y_hzZWGgeVKpA/viewform