Table of Contents

List of Figures

Introduction

For the deaf and hard-of-hearing community, sign language is an essential means of communication that allows them to express themselves and engage with the outside world. However, barriers to communication frequently exist between people who use sign language and others who do not comprehend it. By translating sign language motions into text in real time, our Sign Language Detection App seeks to close this gap. With the use of cutting-edge machine learning and computer vision technology, this software offers a simple and approachable way to ensure smooth communication.

We think taking an initiative like this is very essential in order to promote greater understanding and inclusivity in a society which might not be as accessible to the disabled community as it is to the abled. Hence, enabling instantaneous communication between individuals who utilise sign language and those who do not can greatly minimise misinterpretations and enhance the standard of daily interactions.

Utilizing a Convolutional Neural Network (CNN) for gesture recognition and Natural Language Processing (NLP) for contextual understanding, our approach provides an efficient and user-friendly solution for sign language translation. Currently, our application is only able to detect words like Hello, Thank You and Home, but we intend to keep working on the project to include Indian Sign Language (ISL) alphabets as well as different sign languages.

Additionally, this app is made to be extremely flexible. It can work on different backgrounds without any decrease in the accuracy of the detection of sign language. Moreover, the hardware requirement is just an access to a working camera, thus making the app user-friendly for people of all ages and technological proficiency levels.

The design and working of the model is discussed thoroughly in the methodology section of this report.

Abstract

The project “Sign Language Translator App Using Artificial Intelligence” is aimed at bridging the communication gap between the deaf and sound sensitive people. We harness Artificial Intelligence and Computer Vision to convert sign language gestures into text in real-time. We aim to provide the technical implementation and potential future enhancements to improve accessibility and communication for the deaf people.

The project's main goal is to create an accurate and effective system that can recognise and interpret a wide range of sign language motions. It is a useful tool for a more inclusive society as it can help with various day to day interactions that were difficult before such as customer service, banking or other government mandatory procedures, improve accessibility in public services, as well as improve educational infrastructure.

Problem Statement

We all are aware of the problems faced by the deaf community to simply exist and function on the daily. The methods currently being used are inadequate and inefficient when taken into account how frequent we all need to have conversations. Lip-reading and written notes can only take one so far in terms of communication. There has been an urgent need for the development of an efficient and real-time translation tool which can accurately convert sign language into text.

Some applications do exist online that are used for sign language translation but most of them use images capture to translate the sign language or require specific hardware like green screen, lighting equipment, expensive gloves with markers, calibration tools etc. These lead to high maintenance cost and less availability to people with all types of financial background. Therefore, in our approach we decide to work on these factors, so the only tool required to have a simple conversation is a phone with camera enabling people from around the world to have access to it.

Tools Used

Software

- Jupyter Notebook

  It is an open-source application widely used in data science and machine learning. It allows us to create, save and share live codes in a wide variety of language such as python which was mainly used to create the backend of the project.

- Visual Studio Code

  It is once again a free, open-source code editor that supports a wide range of programming languages and helps in integration of multiple languages in a project. We used VS code to develop and link the front end of the project to the backend.

- Command Prompt

  It is an inbuilt command line interpreter in Windows. It performed a key role in performing administrative tasks such as downloading various dependencies, files and directory management, system configuration etc.

- Flask API

  It is a python specific framework designed for the development of web application. It provides flexible routing to handle different HTTP methods and it's built in development server helped us to create a frontend efficiently.

Hardware

- Webcam

  A primary webcam allowed us to input real-time video feed for data collection as well as the accurate recognition of the signs.

Methodology

For the project, we searched for existing datasets but were unable to locate any in the form of raw photos that complied with the requirements of our software. Hence it was decided, that we would create our own data set.

Our proposed methodology consists of four phases; Data Collection, Preprocessing and feature extraction, Data Cleaning and Labelling, and Gesture Recognition & Translation. Below is a detailed discussion of each of the phases.

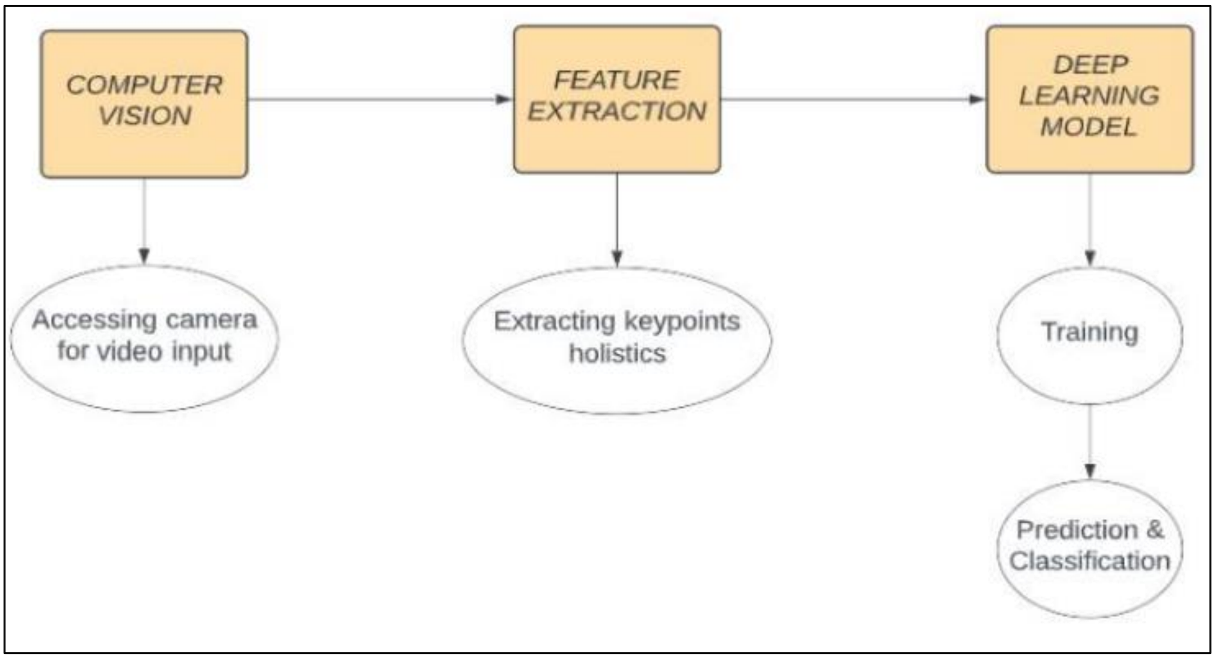

COMPUTER VISION
FEATURE EXTRACTION
DEEP LEARNING MODEL
Accessing camera for video input
Extracting keypoints holistics
Training
Prediction & Classification

Fig. 1 Methodology

Phase 1. Data Collection:

MediaPipe is an open-source framework developed by Google which makes the processing of real time input data easier by providing a customizable and efficient machine-learning pipelines. In this project, we make use of the Face Detection Model and Face Mesh to detect various landmarks on our face and the Pose Landmark Detection Model to extract human body landmarks which includes 33 key points as shown in Fig. 2. Further, Hand Landmark Detections is used to detect 21 key points, shown in Fig. 3, within the hand region.

MediaPipe holistics is an extension of MediaPipe that combines all the above models to allow a more comprehensive analysis.

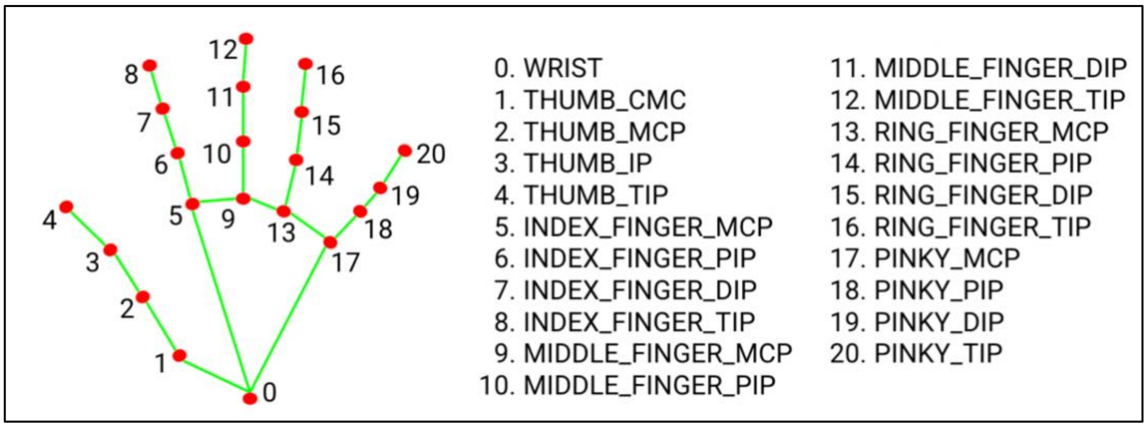


Fig. 2

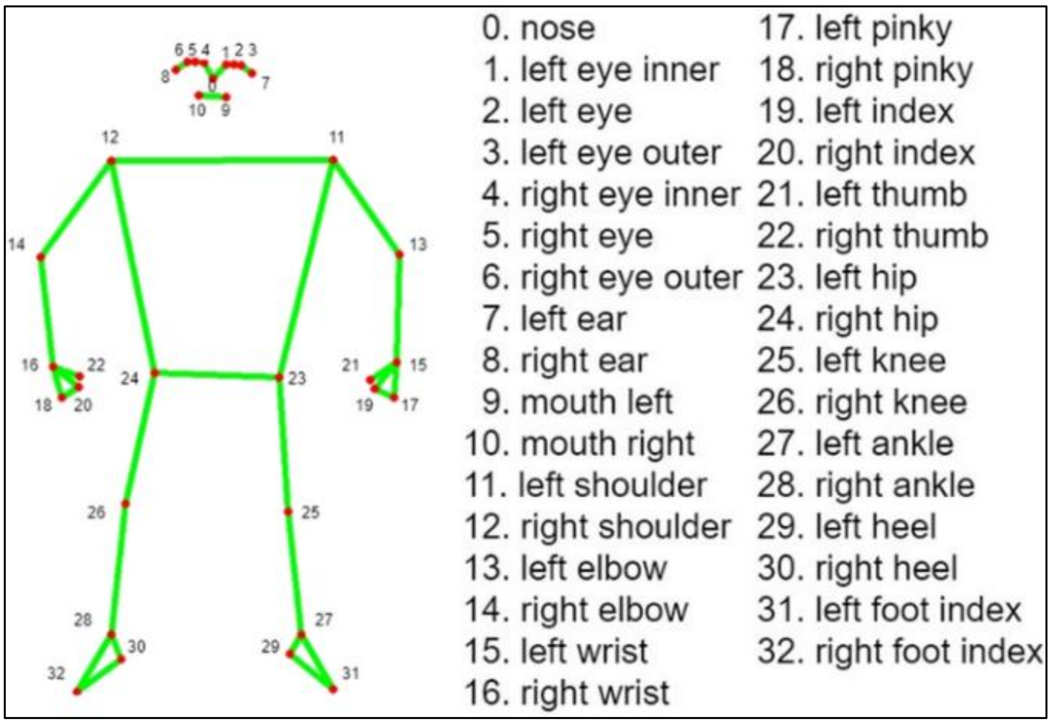


Fig. 3

TensorFlow, another Google framework, provides a wide variety of services. The strong libraries for data processing and augmentation help with the real-time approach and the augmented data help to improve model generalization. The Keras API facilitates the construction of Convolution Neural Networks (CNNs).

OpenCV and NumPy are both important libraries used in the project. NumPy provides an N-dimensional array object called 'ndarray' used in data manipulation, meanwhile, OpenCV is used for capturing, processing and analysing real-time video.

OpenCV and MediaPipe are used to capture the webcam feed using 'cv2.VideoCapture(0)'. A MediaPipe holistics model is initialised with a minimum detection confidence of 0.5 and minimum tracking confidence of 0.5. Here, each frame is read, landmarks are detected and then drawn on the frame using a custom function that specifies different colours for each set of landmarks for easier distinction.

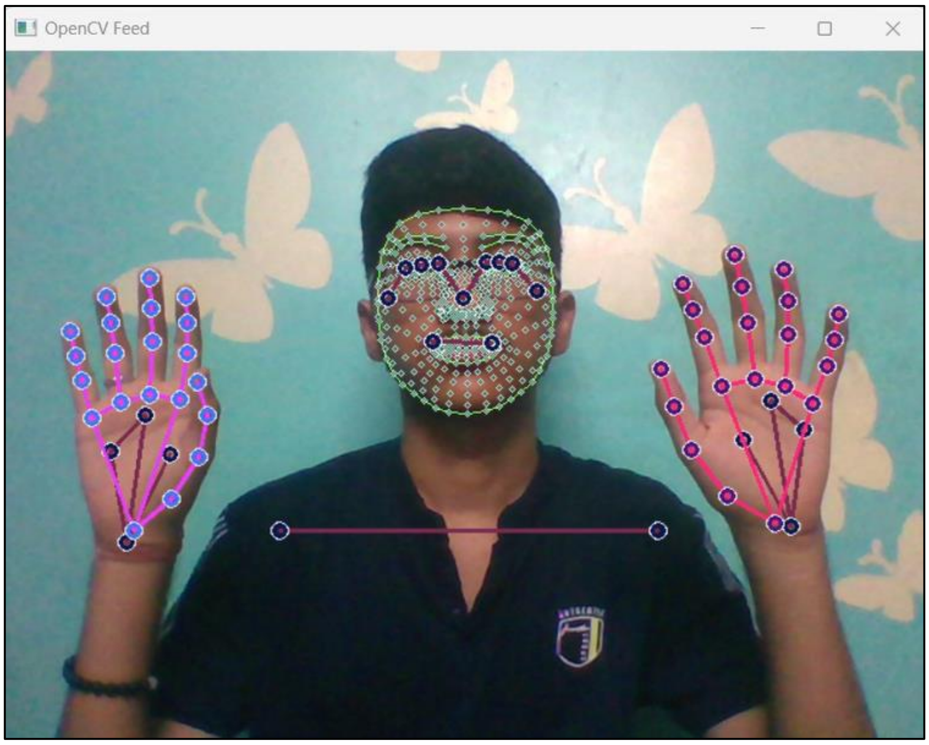


Fig. 4

Three set of words were chosen for the proposed methodology; Hello, Thank You and Home. The gestures are based on Indian Sign Language (ISL).

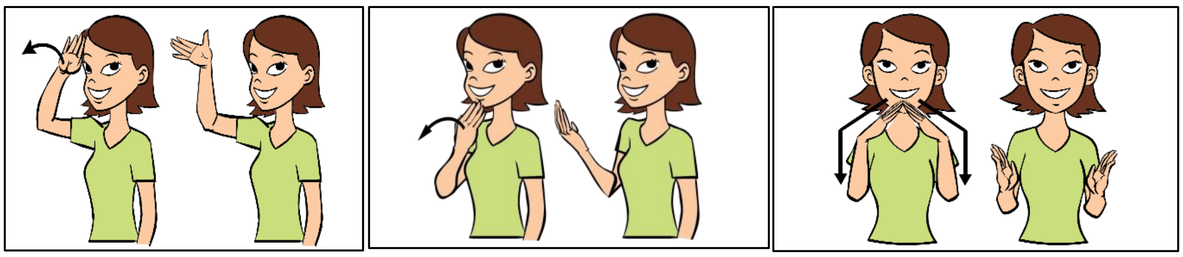


Fig. 5

Phase 2. Data Preprocessing and Feature Extraction:

After collecting the dataset in the form of videos, a total of 100 sequences were collected with each frame having a sequence length of 10. The pipeline provided by MediaPipe Holistics was used to mark and save landmarks for face, hand and pose for each of the frames so as to obtain a wide variety of dataset for a more accurate model. The total collected landmarks came out to be more than 1600.

Phase 3. Data Cleaning and Labelling:

Now, out of 1600 landmarks, there is a possibility that some landmarks are blurry and hence, can lead to incorrect feature detection or no feature detection at all. Further the more noise the data has, there are more chances of model being overfitted toward one word, which leads to loss in accuracy. Therefore, data cleaning is done by fattening, concatenating the landmarks and checking and removing any null entries from the dataset.

Labelling is an important step to keep our code precise and for avoiding any confusion for the model. Therefore, 3 folders are created for each word, each folder consisting of another 100 folders that contain 10 .npy files each.

Phase 4. Gesture Recognition & Translation:

After collecting the dataset and building an LSTM model, we are now capable of using the said model to correctly recognise the different signs in real time and displaying the results on the screen.

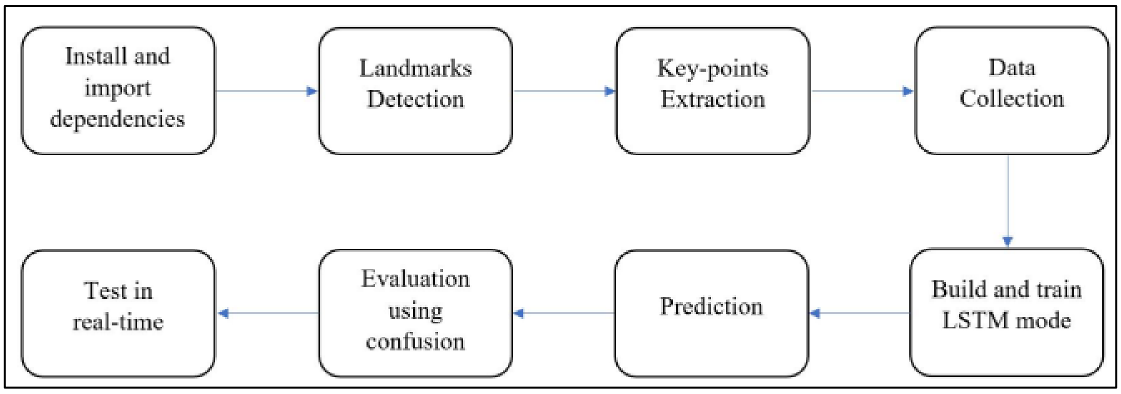
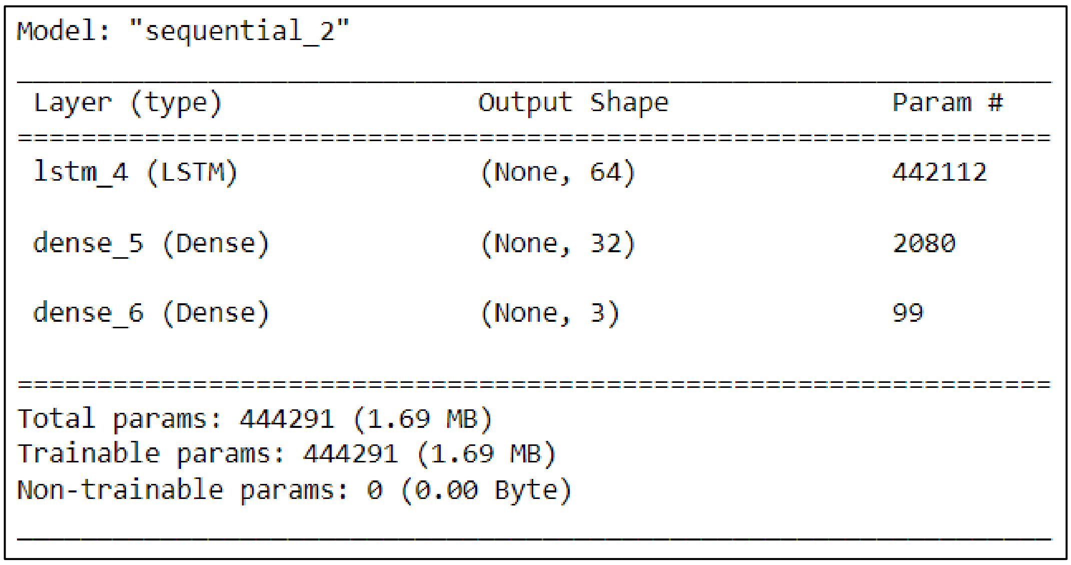

Install and import dependencies
Landmarks Detection
Key-points Extraction
Data Collection
Build and train LSTM mode
Prediction
Evaluation using confusion
Test in real-time

Fig. 7

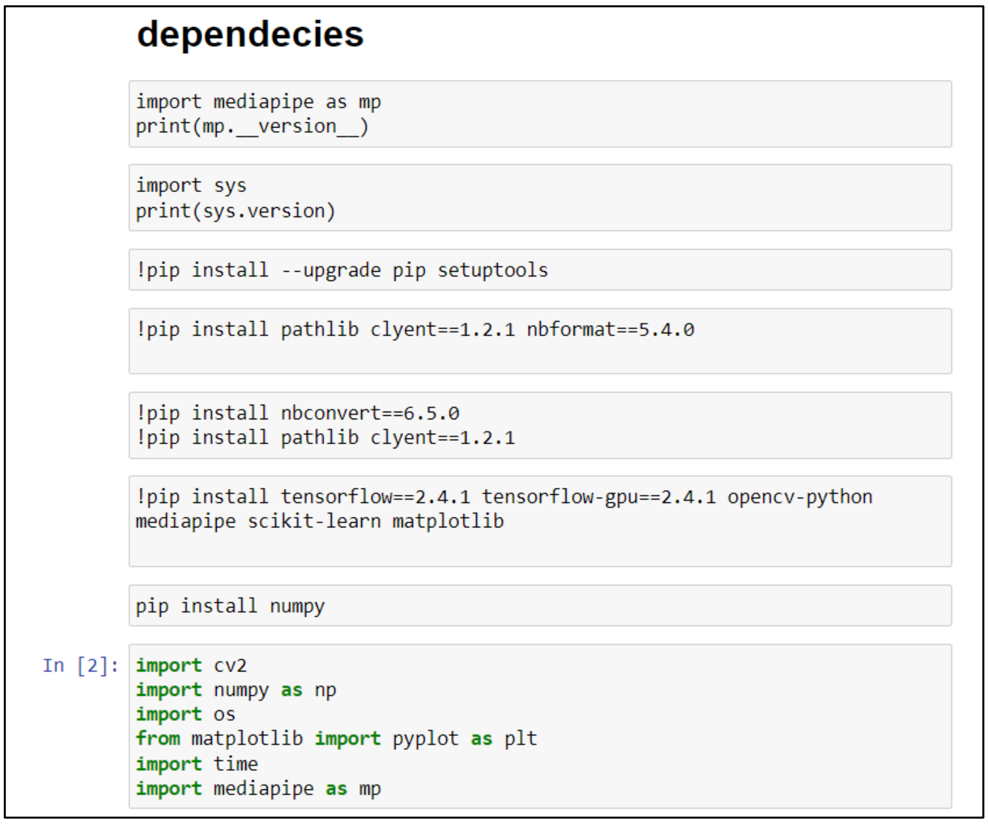


```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_4 (LSTM)               (None, 64)                442112

 dense_5 (Dense)             (None, 32)                2080

 dense_6 (Dense)             (None, 3)                 99

=================================================================
Total params: 444291 (1.69 MB)
Trainable params: 444291 (1.69 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Fig. 8

Model Architecture

From the previous works we analysed, it was found that the key issue faced by the researchers was the non-uniform background which led to a lot of inaccuracy in gesture recognition. We approached this issue by taking input in front of different backgrounds and making use of Google's MediaPipe strong library to only detect landmarks on face, pose and hands.



dependecies

```
import mediapipe as mp
print(mp.__version__)
```

```
import sys
print(sys.version)
```

```
!pip install --upgrade pip setuptools
```

```
!pip install pathlib clyent==1.2.1 nbformat==5.4.0
```

```
!pip install nbconvert==6.5.0
!pip install pathlib clyent==1.2.1
```

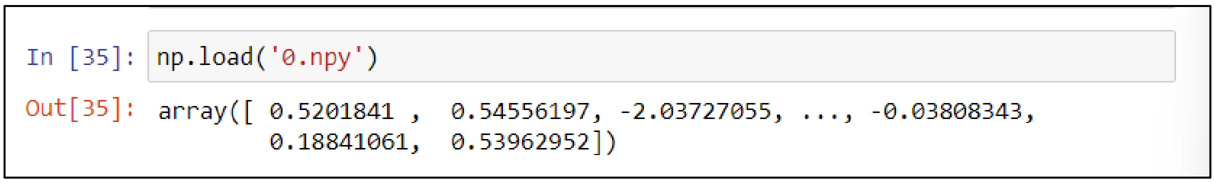
```
!pip install tensorflow==2.4.1 tensorflow-gpu==2.4.1 opencv-python
mediapipe scikit-learn matplotlib
```

```
pip install numpy
```

```
In [2]: import cv2
        import numpy as np
        import os
        from matplotlib import pyplot as plt
        import time
        import mediapipe as mp
```

Fig. 9

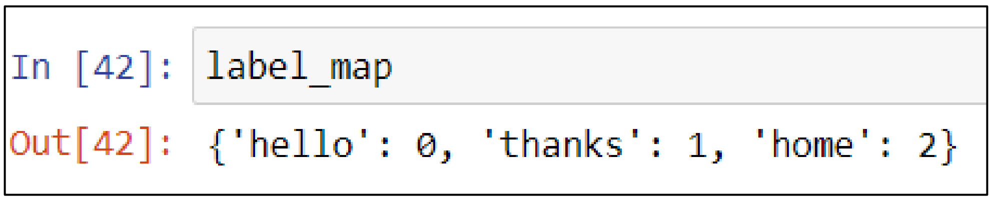
After importing the necessary libraries, the cameras open to capture a video that utilizes a user defined function to draw the landmarks on face, pose, left hand as well as right hand. Next, we extract key points and store them in the form of arrays and save them in a '0.npy' file format.



```
In [35]: np.load('0.npy')

Out[35]: array([ 0.5201841 ,  0.54556197, -2.03727055, ..., -0.03808343,
                 0.18841061,  0.53962952])
```

Fig. 10

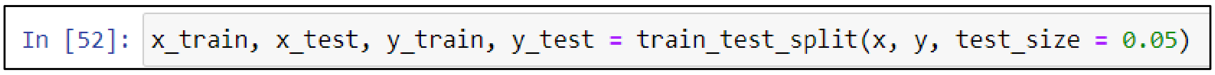
The next step is to create folders and collect key points to make an extensive dataset. Further, ‘sklearn.model_selection’ and ‘tensorflow.keras.utils’ libraries are used to process data and create a label map.



```
In [42]: label_map
Out[42]: {'hello': 0, 'thanks': 1, 'home': 2}
```

Fig. 11

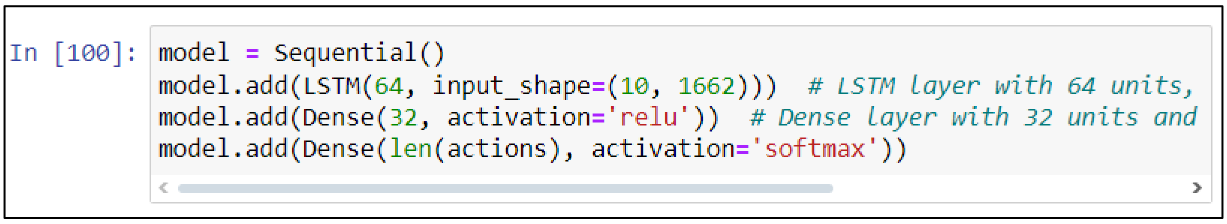
The ‘train_test_split’ from ‘sklearn.model_selection’ is used to split the data into training and testing sets as shown in Fig. 12



```
In [52]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.05)
```

Fig. 12

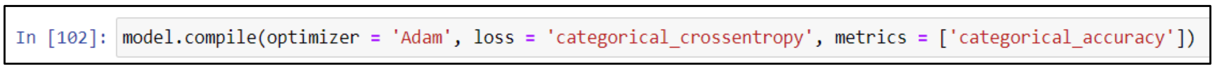
A deep learning model, ‘Sequential_2’, is made using Sequential, LSTM, Dense and TensorBoard modules imported from TensorFlow. The model consists of three layers. The first layer is an LSTM layer with 64 units, followed by two Dense layers one with 32 units and one of the lengths of the ‘actions’ array with a SoftMax function that introduces non-linearity in the model.



```
In [100]: model = Sequential()
          model.add(LSTM(64, input_shape=(10, 1662)))  # LSTM layer with 64 units,
          model.add(Dense(32, activation='relu'))  # Dense layer with 32 units and
          model.add(Dense(len(actions), activation='softmax'))
```

Fig. 13

Since the model we prepared needed an optimiser that could handle massive amounts of data without compromising on efficiency, we chose ‘Adam’. Adaptive Moment Estimation, in short Adam, combines PMSprop and Stochastic Gradient Descent to provide a great optimization algorithm that minimizes loss function during training of neural networks, thereby improving the training speed and reaching convergence faster. One other reason, we opted Adam was of its ability to handle noise.



```
In [102]: model.compile(optimizer = 'Adam', loss = 'categorical_crossentropy', metrics = ['categorical_accuracy'])
```

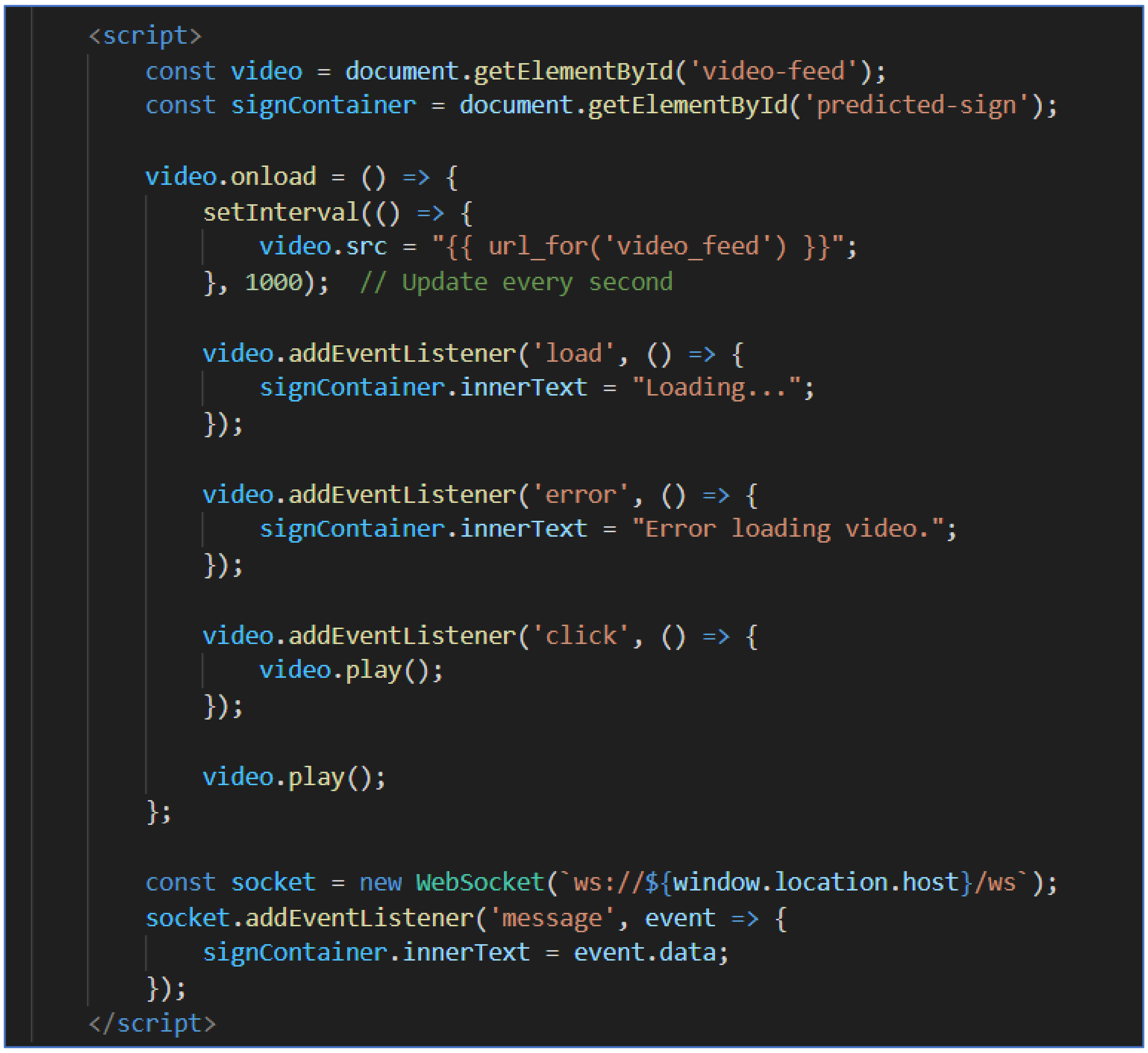
Fig. 14

Lastly, the model is saved in a ‘keras’ format and the weights are loaded. For testing, the background noise threshold is set to 0.6 with the consistency threshold set at 10. It also stores the last 10 predictions to ensure that the decision is made according to the last 10 frames. Then the action is displayed on the frame.

Front End

For making the front end of the project, we divided it into two parts; HTML structure and Flask Application.

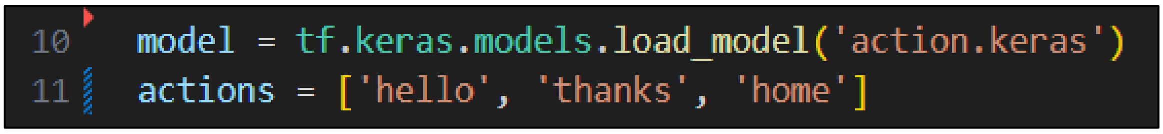
The HTML structure was further divided into the HTML-CSS part and the JavaScript part. The former was used to set up a webpage that uses CSS to design the body and include a container for title, video feed, and an area for showing the predicted sign language. JavaScript was used to establish a web socket connection and to continuously update the video feed for the real time prediction.



```
<script>
    const video = document.getElementById('video-feed');
    const signContainer = document.getElementById('predicted-sign');

    video.onload = () => {
        setInterval(() => {
            video.src = "{{ url_for('video_feed') }}";
        }, 1000);  // Update every second

        video.addEventListener('load', () => {
            signContainer.innerText = "Loading...";
        });

        video.addEventListener('error', () => {
            signContainer.innerText = "Error loading video.";
        });

        video.addEventListener('click', () => {
            video.play();
        });

        video.play();
    };

    const socket = new WebSocket(`ws://${window.location.host}/ws`);
    socket.addEventListener('message', event => {
        signContainer.innerText = event.data;
    });
</script>
```

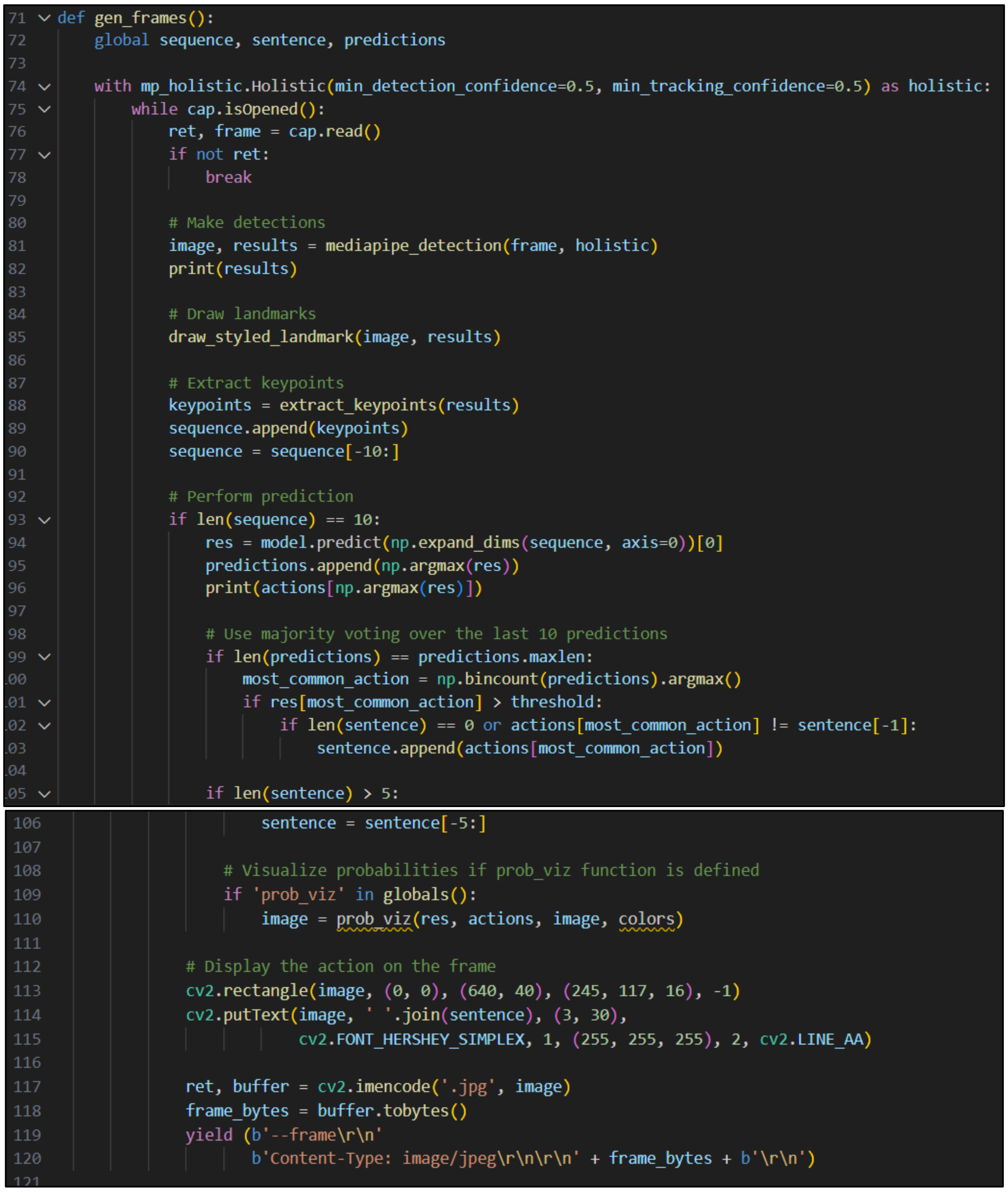
Fig. 15

The Flask API and its modules such as Flask, render_template and Response were used to create the backend of the webpage. After importing the necessary libraries and redefining the functions from our backend, it loads our pre-trained model.



```python
model = tf.keras.models.load_model('action.keras')
actions = ['hello', 'thanks', 'home']
```

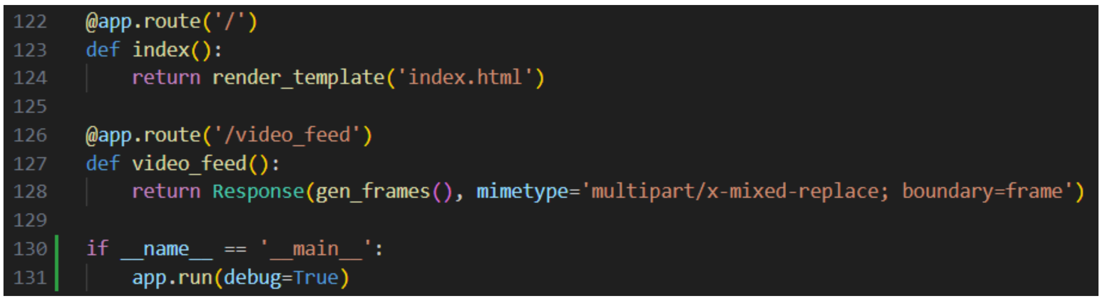
Fig. 16



```python
def gen_frames():
    global sequence, sentence, predictions

    with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
        while cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                break

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

            # Draw landmarks
            draw_styled_landmark(image, results)

            # Extract keypoints
            keypoints = extract_keypoints(results)
            sequence.append(keypoints)
            sequence = sequence[-10:]

            # Perform prediction
            if len(sequence) == 10:
                res = model.predict(np.expand_dims(sequence, axis=0))[0]
                predictions.append(np.argmax(res))
                print(actions[np.argmax(res)])

                # Use majority voting over the last 10 predictions
                if len(predictions) == predictions.maxlen:
                    most_common_action = np.bincount(predictions).argmax()
                    if res[most_common_action] > threshold:
                        if len(sentence) == 0 or actions[most_common_action] != sentence[-1]:
                            sentence.append(actions[most_common_action])

                if len(sentence) > 5:
                    sentence = sentence[-5:]

                # Visualize probabilities if prob_viz function is defined
                if 'prob_viz' in globals():
                    image = prob_viz(res, actions, image, colors)

            # Display the action on the frame
            cv2.rectangle(image, (0, 0), (640, 40), (245, 117, 16), -1)
            cv2.putText(image, ' '.join(sentence), (3, 30),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

            ret, buffer = cv2.imencode('.jpg', image)
            frame_bytes = buffer.tobytes()
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame_bytes + b'\r\n')
```

Fig. 17

The ‘gen_frames’ function reads the frame from the live feed through the web cam, detect the landmarks, extract key points and make predictions in real time. Finally, the Flask application is connected to the HTML counterpart and is run in debug mode.



```
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/video_feed')
def video_feed():
    return Response(gen_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')

if __name__ == '__main__':
    app.run(debug=True)
```

Fig. 18

The interface of the webpage is displayed. It has a container to indicate the signs detected and just below, the central part displays a live video feed from the webcam. The landmarks seen in the photo indicates that the tracking is in progress and is ready to recognise signs.

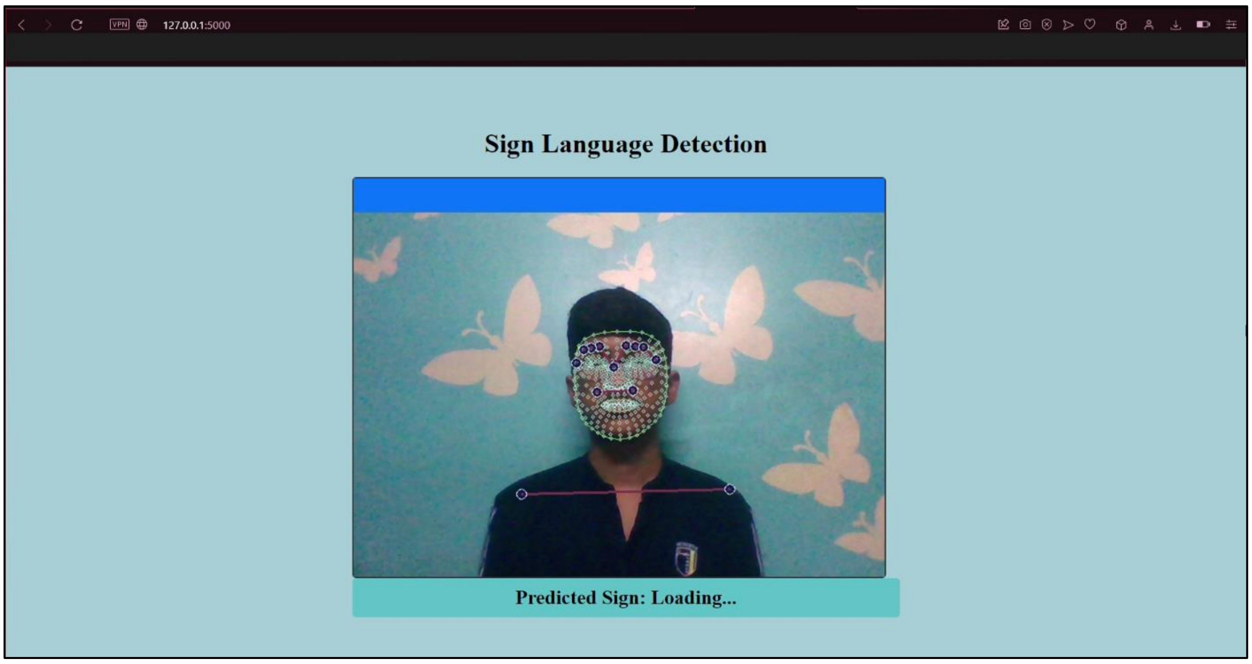
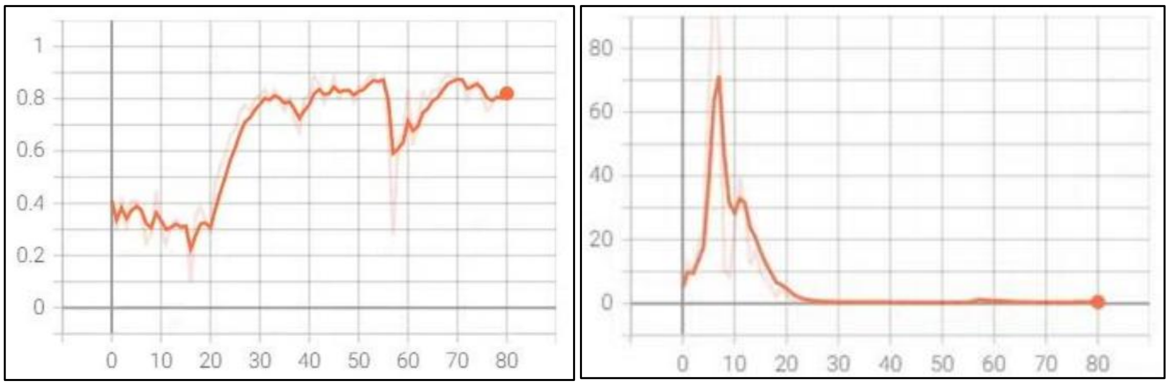


Fig. 19

Results and Discussions

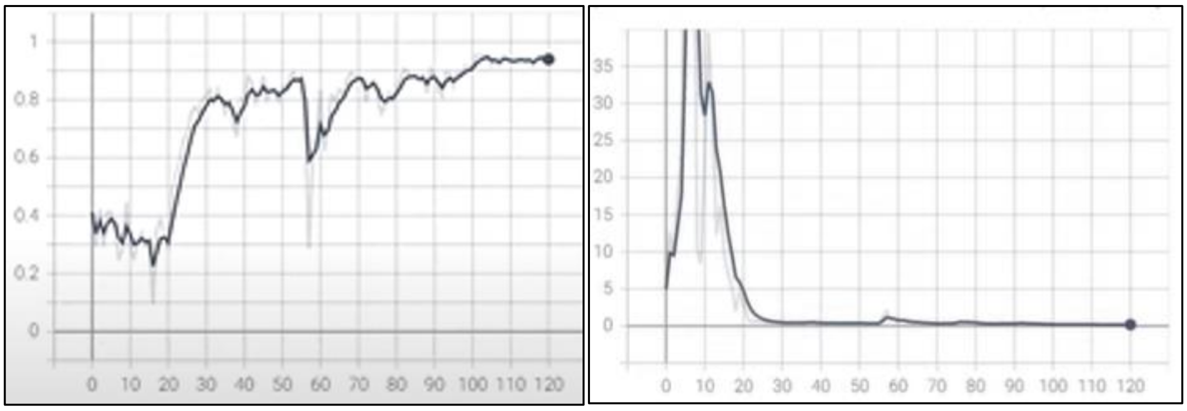
The collected dataset was split into training and testing data. The evaluation metric used was how accurately the sings were being recognised by the model.



Epoch Categorial Accuracy Epoch Loss

Fig. 20

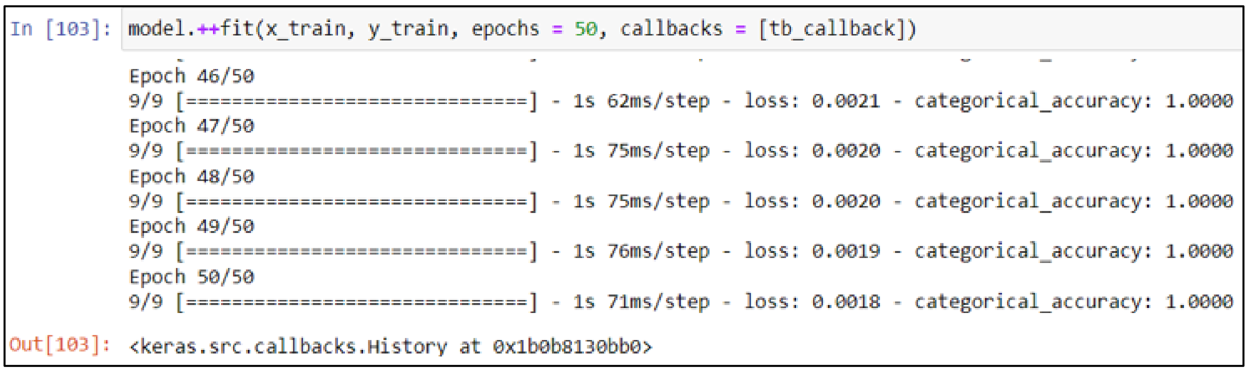
Initially, the model was working successfully with 100% accuracy but due to large number of epochs and the consequent overtraining of the data, the accuracy score dropped to 80%. It was observed that the model developed bias or overfitted to a particular word. Hence, the epochs were reduced and relevant changes were made to fix this issue.



Epoch Categorial Accuracy Epoch Loss

Fig. 21

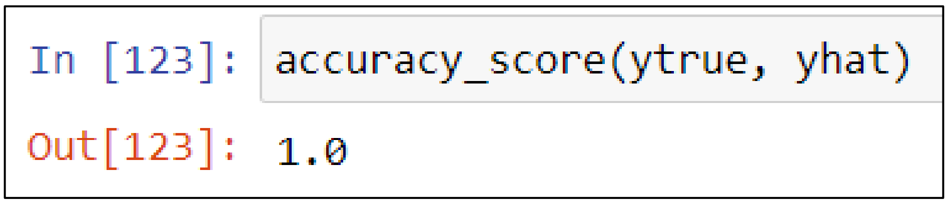
The accuracy was now observed to be 100% and the model is recognising the camera input correctly.



```
In [103]: model.++fit(x_train, y_train, epochs = 50, callbacks = [tb_callback])
          Epoch 46/50
          9/9 [==============================] - 1s 62ms/step - loss: 0.0021 - categorical_accuracy: 1.0000
          Epoch 47/50
          9/9 [==============================] - 1s 75ms/step - loss: 0.0020 - categorical_accuracy: 1.0000
          Epoch 48/50
          9/9 [==============================] - 1s 75ms/step - loss: 0.0020 - categorical_accuracy: 1.0000
          Epoch 49/50
          9/9 [==============================] - 1s 76ms/step - loss: 0.0019 - categorical_accuracy: 1.0000
          Epoch 50/50
          9/9 [==============================] - 1s 71ms/step - loss: 0.0018 - categorical_accuracy: 1.0000
Out[103]: <keras.src.callbacks.History at 0x1b0b8130bb0>
```

Fig. 22

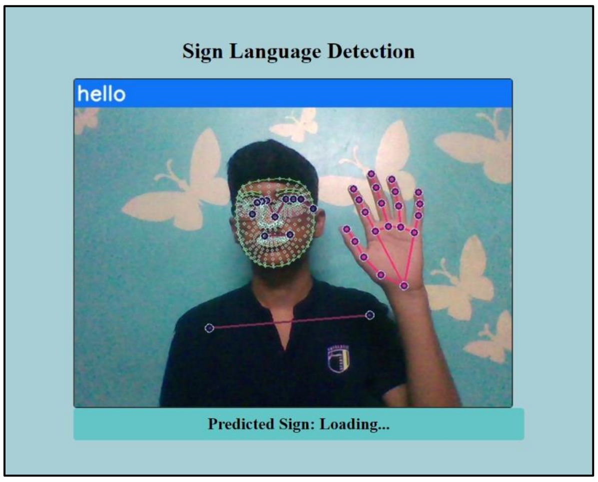
The test model’s accuracy also came out to be 1.0.



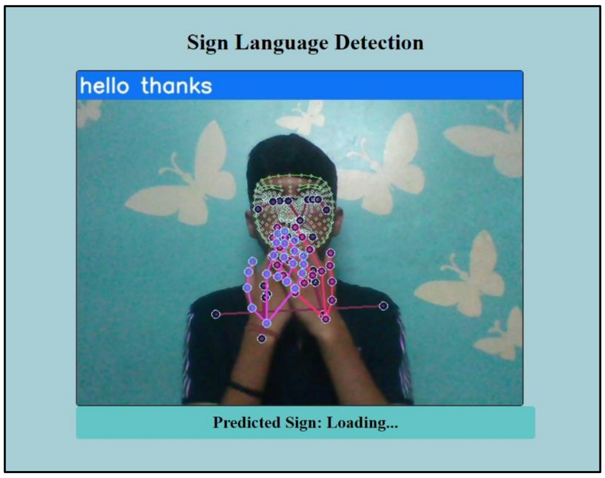
```
In [123]: accuracy_score(ytrue, yhat)
Out[123]: 1.0
```

Fig. 23

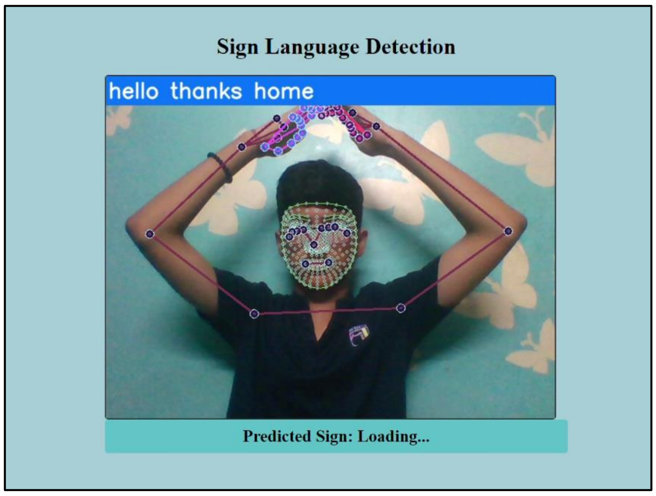
When tested in real life, the model detected the signs without any error. The following snapshots support the previous statement.



Hello



Thank You



Home

Fig. 24

Conclusion

In this project, we used Google’s MediaPipe and LSTM to develop a system to recognise and translate Indian Sign Language (ISL). We collected an extensive dataset which was then pre-processed, augmented, extracted, cleaned and labelled before being fed to the model to be trained. The network achieved an accuracy of 100% on the testing data. Finally, the system was tested with real-time data fed into the model, that displayed results for each gesture. Finally, we successfully developed a front-end website for the model that works on the local host server. In conclusion, we learnt following different approaches and being diligent in figuring out a solution is key in any project development. We learnt the approach to overcome the challenges that arise while working towards a bigger goal.

Future Work

- We can develop a better and complete model that incorporates the alphabet as well as other words used by the sign language community.
- Integration of different types of sign language is also one of our future goals so as to increase inclusivity.
- Finally, the deployment of the project and making it available to mobile applications in an offline mode is the end target.

References

1. Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E., Hays, M., ... & Grundmann, M. (2019). Mediapipe: A framework for building perception pipelines. *arXiv preprint arXiv:1906.08172*.

2. Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E., Hays, M., ... & Grundmann, M. (2019, June). Mediapipe: A framework for perceiving and processing reality. In *Third workshop on computer vision for AR/VR at IEEE computer vision and pattern recognition (CVPR)* (Vol. 2019).

3. Singh, A. K., Kumbhare, V. A., & Arthi, K. (2021, June). Real-time human pose detection and recognition using mediapipe. In *International conference on soft computing and signal processing* (pp. 145-154). Singapore: Springer Nature Singapore.

4. Veluri, R. K., Sree, S. R., Vanathi, A., Aparna, G., & Vaidya, S. P. (2022, March). Hand gesture mapping using MediaPipe algorithm. In *Proceedings of Third International Conference on Communication, Computing and Electronics Systems: ICCCES 2021* (pp. 597-614). Singapore: Springer Singapore.

5. Kothadiya, D., Bhatt, C., Sapariya, K., Patel, K., Gil-González, A. B., & Corchado, J. M. (2022). Deepsign: Sign language detection and recognition using deep learning. *Electronics*, *11*(11), 1780.

6. Moryossef, A., Tsochantaridis, I., Aharoni, R., Ebling, S., & Narayanan, S. (2020). Real-time sign language detection using human pose estimation. In *Computer Vision–ECCV 2020 Workshops: Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16* (pp. 237-248). Springer International Publishing.

7. Sanmitra, P. R., Sowmya, V. S., & Lalithanjana, K. (2021). Machine Learning Based Real Time Sign Language Detection. *International Journal of Research in Engineering, Science and Management*, *4*(6), 137-141.

8. M. Liu, M. A. A. Murad, J. Chen and F. Milano, "Modeling of Protective Relays for Transient Stability Analysis," *2020 IEEE Power & Energy Society General Meeting (PESGM)*, Montreal, QC, Canada, 2020, pp. 1-5, doi: 10.1109/PESGM41954.2020.9281555.

9. Nair, A. V., & Bindu, V. (2013). A review on Indian sign language recognition. *International journal of computer applications*, *73*(22).

10. Zeshan, U., Vasishta, M. N., & Sethna, M. (2005). Implementation of Indian Sign Language in educational settings. *Asia Pacific Disability Rehabilitation Journal*, *16*(1), 16-40.

11. Tomkins, W. (1969). *Indian sign language* (Vol. 92). Courier Corporation.

12. Mariappan, H. M., & Gomathi, V. (2019, February). Real-time recognition of Indian sign language. In *2019 international conference on computational intelligence in data science (ICCIDS)* (pp. 1-6). IEEE.