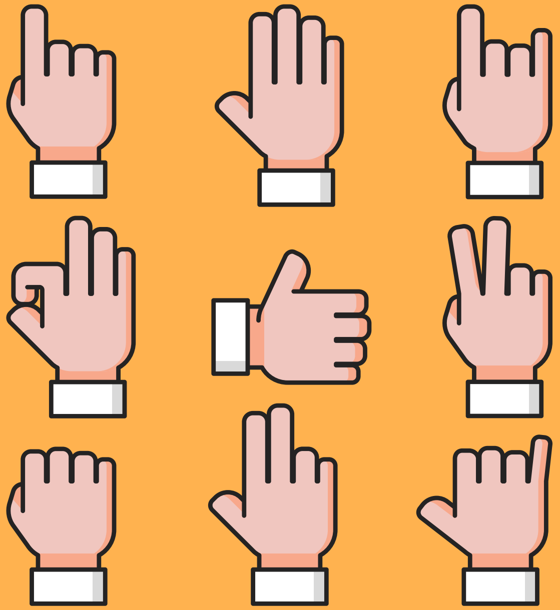
SIGN LANGUAGE DETECTION

Rahul Barodia B20CS047
Agnibha Burman Roy B20CS099

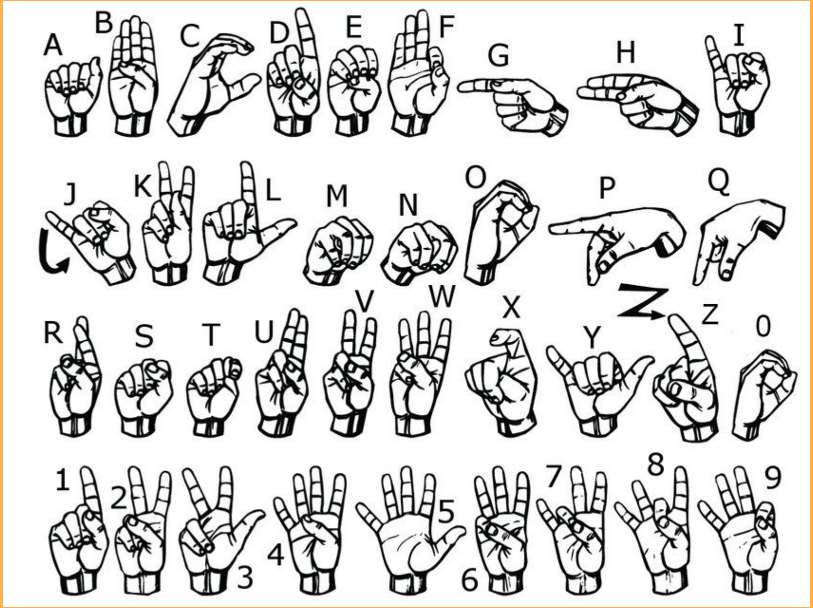
INTRODUCTION

Sign language recognition is a technology that employs computer vision and machine learning to analyze and interpret visual data of sign language gestures. Using algorithms and deep neural networks, the system recognizes and classifies specific signs, translating them into spoken or written language in real-time. This assists communication between individuals who use sign language and those who may not understand it, with applications in assistive technology, education, and inclusive interfaces



COMMONLY USED SIGN LANGUAGES

American Sign Language (ASL) is a natural and visual-gestural language used primarily by the Deaf and hard-of-hearing community in the United States and parts of Canada. It is a complete and complex language with its own grammar, syntax, and vocabulary. ASL is not based on English; it is a distinct and independent language with its own linguistic structure.



FLOW OF OUR PROJECT

Model 1: SVM

Captures hand gestures via webcam, collects image data, and employs MediaPipe for hand landmark extraction. Using SVM to train the model and enabling real-time detection

Model 2 : LSTM

Captures the spatial arrangement of frames which are feeded into a lstm model which predicts the gesture

Model 3 : LSTM to decode from video sequences

Used MediaPipe to detect holistic keypoints LSTM layers to decode sign language gestures from video sequences

MODEL 1 : SVM

Collect_imgs.py

Imported necessary libraries, including os for file operations and cv2 for computer vision tasks.

Created a directory to store the collected images for each class.

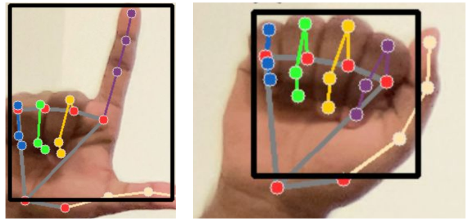
Opened a webcam feed start capturing images for each class. Saved the images in the corresponding class directory.

Create_dataset.py

Extracted hand landmarks using the MediaPipe Hands model.

For each image, a numpy array stores the normalized coordinates of 21 hand keypoints.

If hand landmarks are detected, it flattens the 2D coordinates into a 1D array and appends it to the data list along with the corresponding class label.

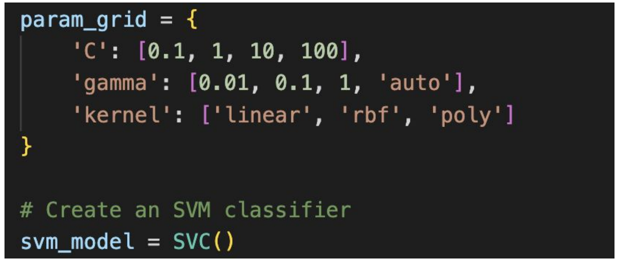


MODEL 1 : SVM

Train_classifier.py :

It loads a dataset containing hand landmarks from images, splits it into training and testing sets, and performs hyperparameter tuning using GridSearchCV.

The best hyperparameters are then used to train a new SVM model.



```
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [0.01, 0.1, 1, 'auto'],
    'kernel': ['linear', 'rbf', 'poly']
}

# Create an SVM classifier
svm_model = SVC()
```

Inference_classifier.py :

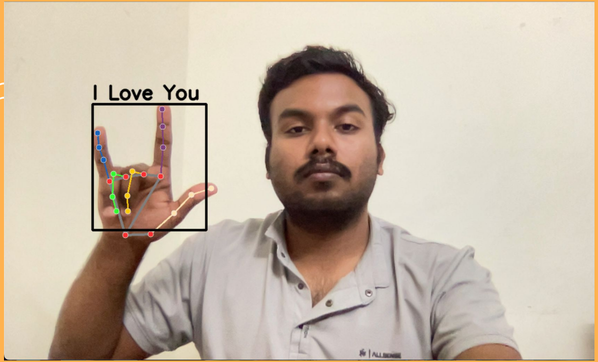
The webcam feed is captured using OpenCV.

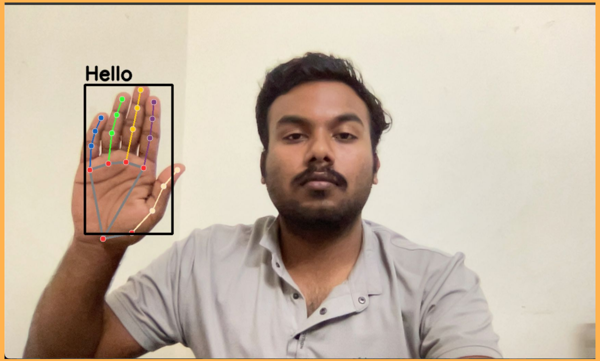
MediaPipe Hands is employed to detect hand landmarks in the video frames.

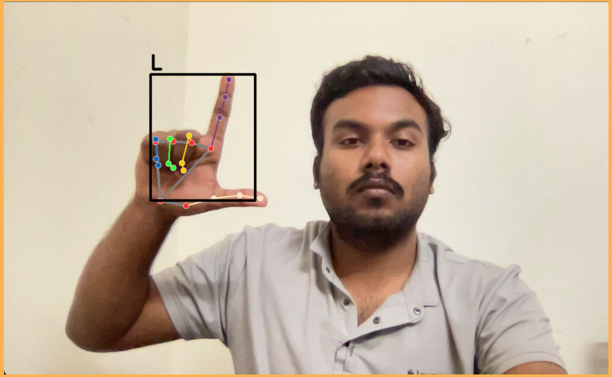
The landmarks' coordinates are normalized and fed into the pre-trained SVM model for prediction.

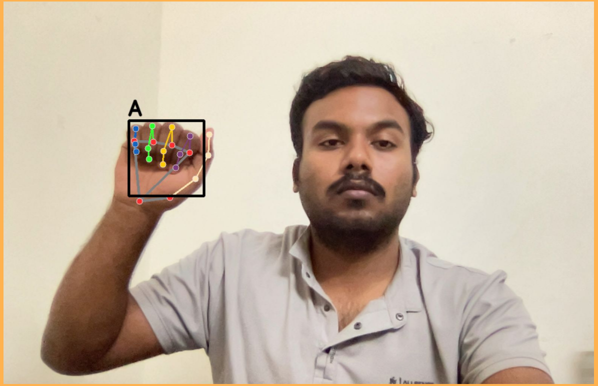
The predicted sign is then displayed on the video feed,

MODEL 1 : SVM









MODEL 2 : LSTM

Introduction

- The Keypoint Classifier project aims to develop a machine learning model using Long Short-Term Memory (LSTM) networks to classify keypoints based on (x, y) coordinates.
- The model is trained on a dataset containing sequences of keypoints, and TensorFlow Lite is employed for deployment on resource-constrained devices.

MODEL 2 : LSTM

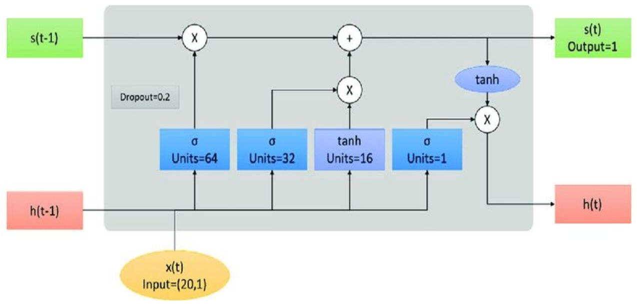
Dataset

- The dataset we prepared consists of (x, y) coordinates representing key points which are captured using the mediapipe library.
- Each sequence is associated with a class label, defining the type of keypoint.
- The dataset is split into training and testing sets using the train_test_split function.

MODEL 2 : LSTM

Model Architecture

- **Input Layer**: Reshape layer to match the LSTM input shape (21 sequences with 2 coordinates each).
- **LSTM Layer:** 20 units with ReLU activation to capture temporal dependencies.
- **Dropout Layer:** 40% dropout rate to prevent overfitting.
- **Dense Layers:** Two dense layers with ReLU and softmax activations for classification.



MODEL 2 : LSTM

Model Compilation

- **Optimizer: Adam optimizer**,it is an efficient and adaptive optimization algorithm that adjusts learning rates for each parameter individually, allowing for faster convergence during neural network training.
- **Loss Function: Sparse categorical cross entropy**,it is a loss function commonly used in the training of neural networks for classification tasks where the classes are mutually exclusive (each input sample can belong to only one class). It is particularly useful when dealing with a large number of classes.
- **Metrics**: **Accuracy**.

MODEL 2 : LSTM

Model Training

- The model is trained on the training set for a maximum of 1000 epochs with a batch size of 128.
- ModelCheckpoint and Early Stopping callbacks are used for saving the best model and preventing overfitting.

MODEL 2 : LSTM

Model Evaluation

- The trained model is evaluated on the test set, and validation loss and accuracy are reported.
- Inference tests are performed on both the original and TensorFlow Lite models.
- A confusion matrix is generated to analyze the model's performance, including precision, recall, and F1-score.

MODEL 2 : LSTM

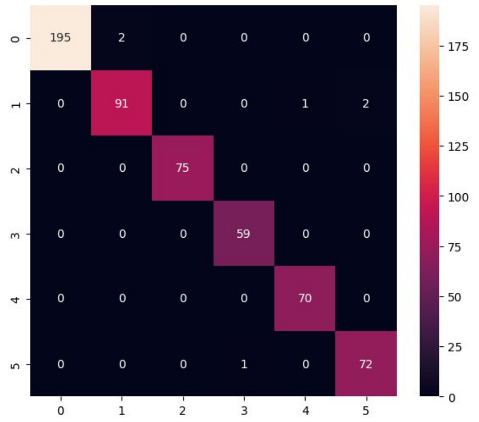
TensorFlow Lite Model Conversion

- The best model is saved and converted to TensorFlow Lite format for deployment.
- **Quantization** is applied to reduce the model size and improve inference speed.
- **Quantization** involves reducing the number of bits used to represent the weights and activations of the network. This is done to make the model more efficient in terms of memory usage and computational resources.

MODEL 2 : LSTM

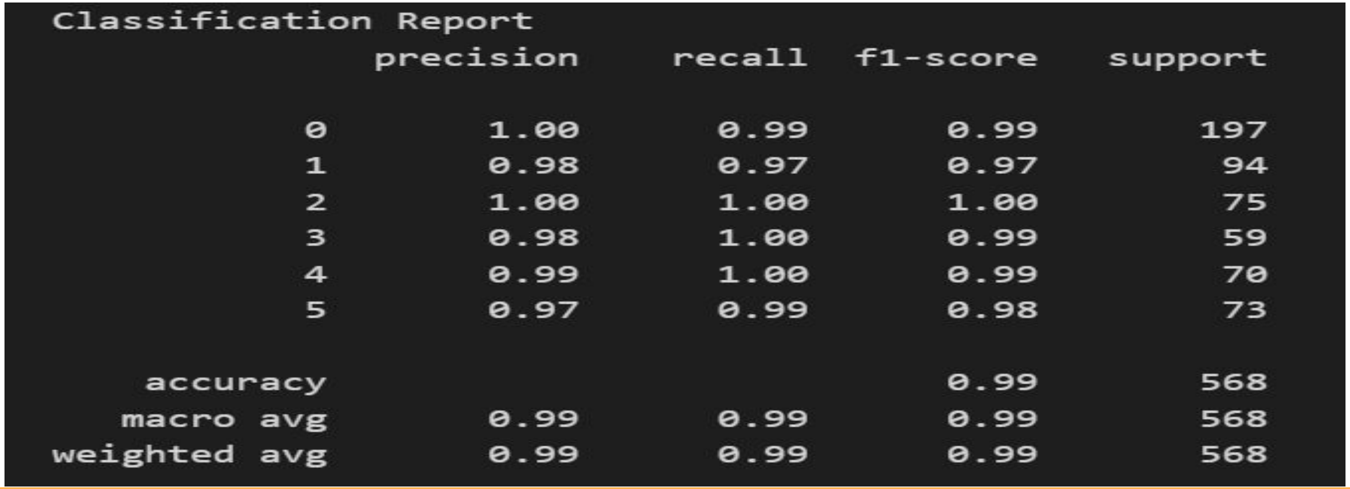
Results and Conclusion

- The model achieves high accuracy on the test set.
- The confusion matrix provides insights into specific misclassifications.
- The TensorFlow Lite model maintains reasonable accuracy with reduced model size.
- The project demonstrates the successful application of LSTM networks for keypoint classification, with potential applications in real-time systems.



MODEL 2 : LSTM

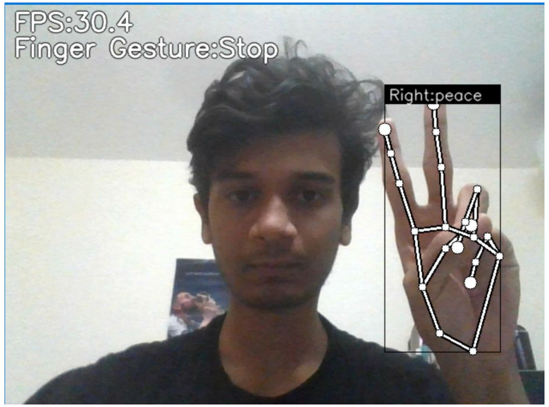
Results and Conclusion

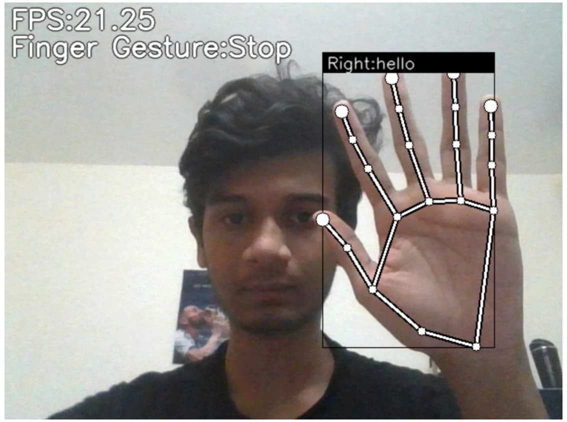


```
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       197
           1       0.98      0.97      0.97        94
           2       1.00      1.00      1.00        75
           3       0.98      1.00      0.99        59
           4       0.99      1.00      0.99        70
           5       0.97      0.99      0.98        73

    accuracy                           0.99       568
   macro avg       0.99      0.99      0.99       568
weighted avg       0.99      0.99      0.99       568
```

MODEL 2 : LSTM

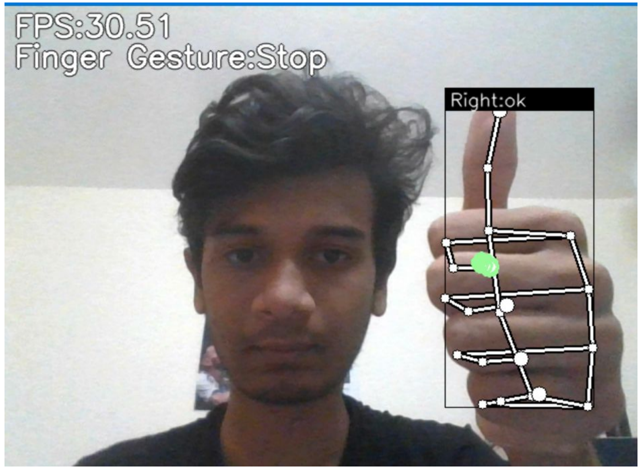
Results and Conclusion

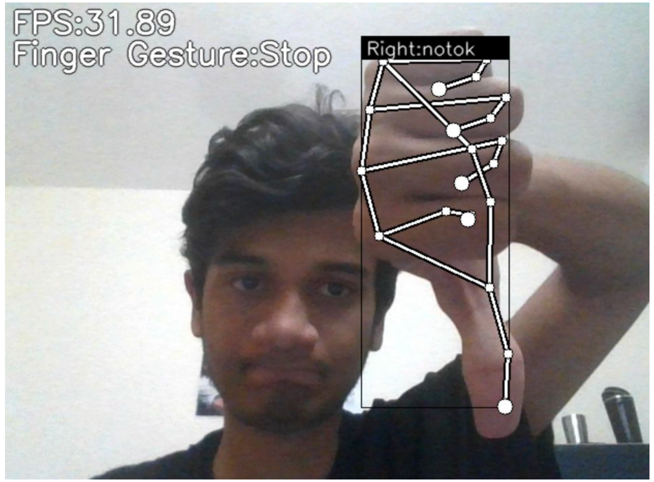




MODEL 2 : LSTM

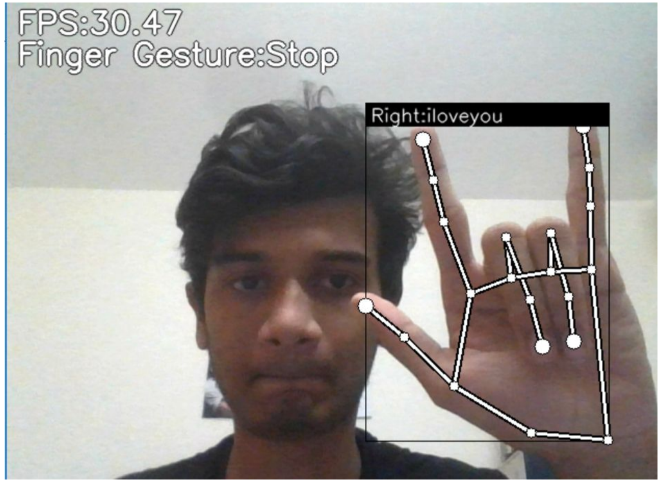
Results and Conclusion

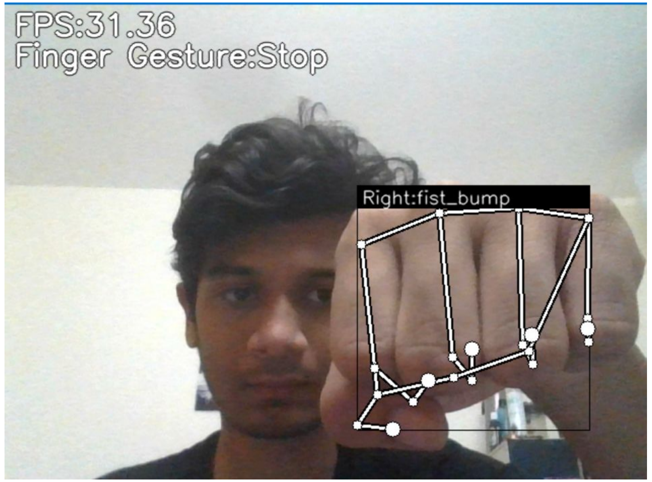




MODEL 2 : LSTM

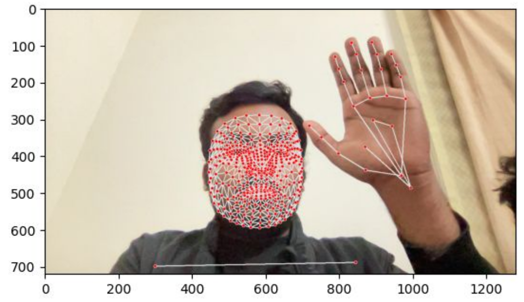
Results and Conclusion



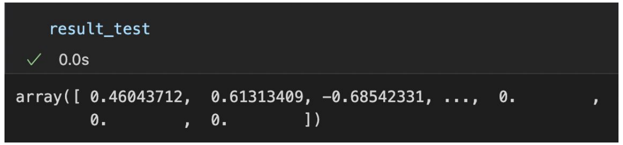


MODEL 3 : LSTM to decode from video sequences

We set up MediaPipe for holistic detection, which includes face, pose, and hand landmarks. OpenCV is used to capture video frames and display the real-time feed.
We extract landmarks from different parts of the body (pose, face, left hand, right hand) and visualize them using matplotlib.



We extracted keypoint information from various landmarks (pose, face, left hand, right hand) detected in an image.
For each landmark type, we collect the x, y, z coordinates and visibility information of each point. If landmarks are not detected, we pad with zeros.
The function extract_keypoints does the same, creating a concatenated array of all these landmarks. The resulting keypoints are saved and loaded using NumPy for further use.



```
result_test
✓ 0.0s

array([ 0.46043712,  0.61313409, -0.68542331, ...,  0.        ,
        0.        ,  0.        ])
```

MODEL 3 : LSTM to decode from video sequences

We defined three sign language actions: 'hello,' 'thanks,' and 'iloveyou.'

Each action is represented by 30 sequences, and each sequence comprises 30 frames.

The data is organized into folders for each action and sequence, with keypoints extracted using the previously implemented MediaPipe library.

In the real-time data collection, our video loop captures frames from a webcam, identifies holistic keypoints using MediaPipe, and visually annotates them.

These annotated keypoints are then exported into NumPy arrays, forming the dataset for training our model.

MODEL 3 : LSTM to decode from video sequences

LSTM :

The model consists of three LSTM layers, each with specific configurations, and three dense layers for high-level reasoning.
The model is compiled with the Adam optimizer and categorical crossentropy loss.

Training is performed for 2000 epochs on the provided training data

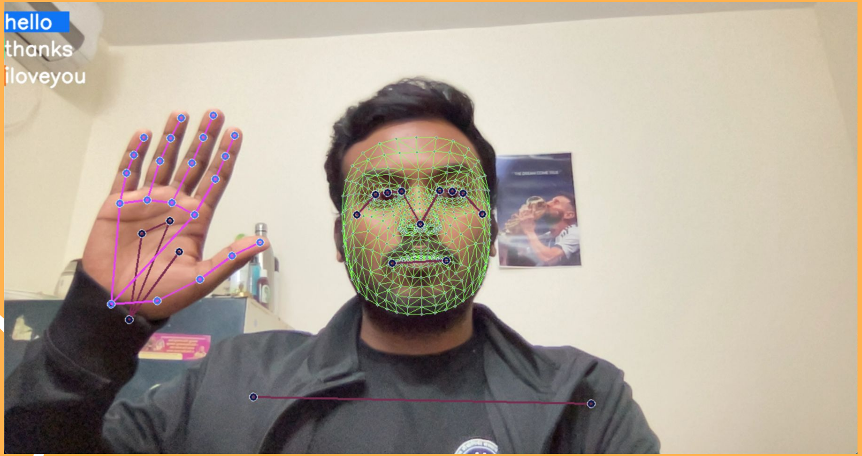
Keypoints are extracted from the detected landmarks, and the last 30 frames' worth of keypoints are maintained in a sequence.

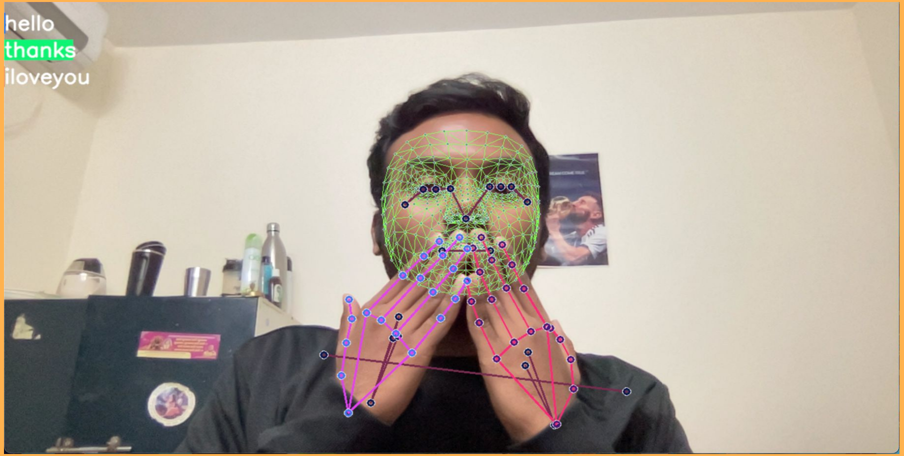
If the sequence reaches 30 frames, it is fed into the model for prediction.

If the predicted class remains consistent in the last 10 predictions and surpasses a defined threshold, it is considered a valid prediction.

The recognized action is added to the sentence list.

MODEL 3 : LSTM to decode from video sequences





THANKS