Sign Language Detection using ACTION RECOGNITION with PYTHON

Ankana Mukherjee	(2105604)
Khushi Kumari	(21052001)
Harsh Kumar Verma	(21051054)
Akash Sharma	(2105006)
Shubhika Kashyap Ojha	(2105243)

INTRODUCTION :

The Sign Language Recognition Using Python project aims to develop a computer vision system capable of recognizing and interpreting sign language gestures through the use of Python programming language and various machine learning techniques. The project focuses on bridging the communication gap between the hearing-impaired community and the general public by enabling real-time translation of sign language gestures into written or spoken language.
.

The system will utilize a webcam or a camera to capture sign language gestures performed by the user. These gestures will then be processed and classified using machine learning algorithms to identify the corresponding words or phrases they represent. The final output will be displayed on the screen or communicated through text-to-speech functionality.

PROJECT OBJECTIVES

Objects :

1. Skill Enhancement
2. Practical Experience
3. Understanding Libraries
4. Project Development
5. Team Collaboration

Project Definition :

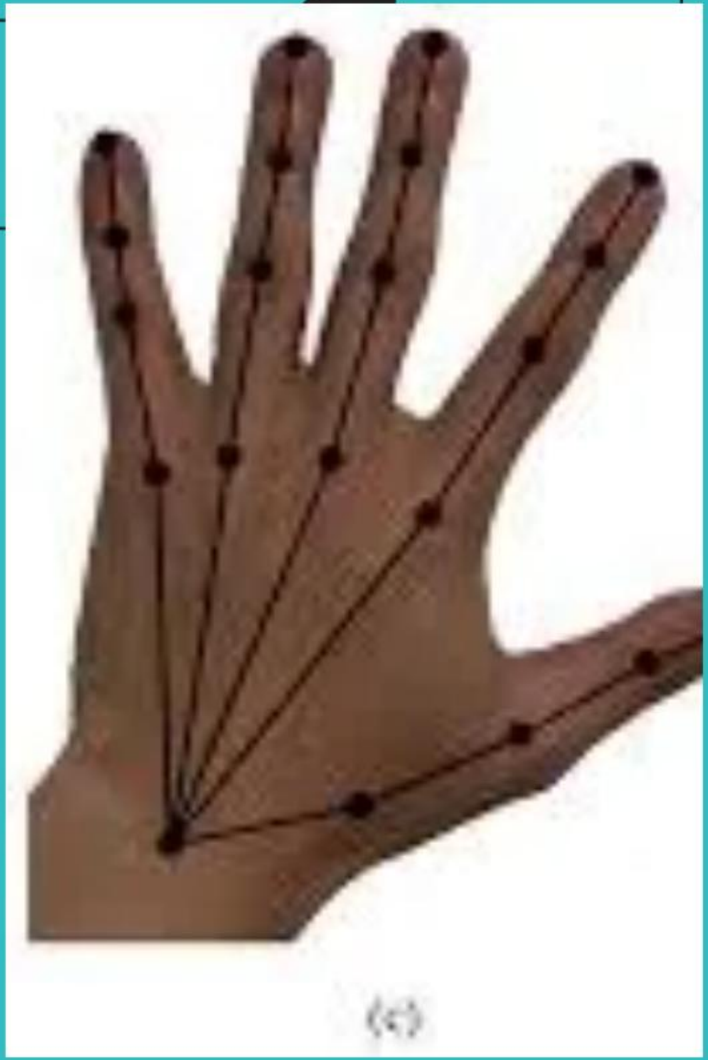
1. Data Collection
2. Data Preprocessing
3. Feature Extraction
4. Machine Learning Models

Project Scope :

1. Real-Time Recognition
2. Basic Vocabulary
3. Single-Handed Gestures
4. Python programming

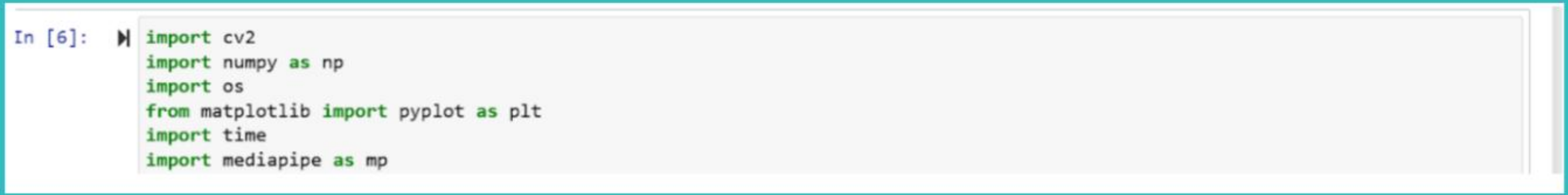
Methodology

We will be presenting the various steps which we followed in this project:



1.Dependencies were installed and imported

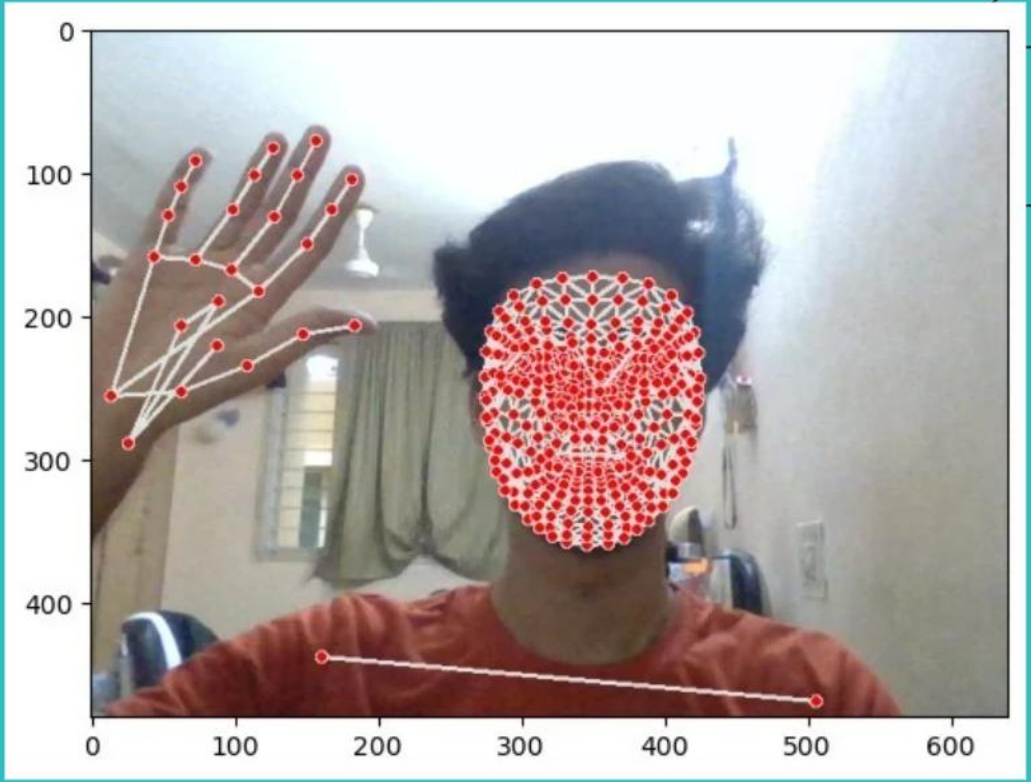
TensorFlow was installed based on the operating system and environment (CPU/GPU)following the official guide. Necessary libraries like TensorFlow, NumPy, OpenCV (optional for video processing), and any other libraries for data manipulation or visualization were imported within the Python code.



```
In [6]:    import cv2
           import numpy as np
           import os
           from matplotlib import pyplot as plt
           import time
           import mediapipe as mp
```

2. Face, Hand, and Pose Landmarks Were Detected (Optional)

This step focused on identifying relevant regions in the video frames. Techniques like Haar cascades, pre-trained models, OpenCV, or MediaPipe libraries might have been used to:
Locate the face in the video frame.
Detect hands within the frame.
Estimate body and hand pose landmarks (key points) for more detailed information.
The sign language video dataset was divided into training, validation, and testing sets.



3. Key Points Were Extracted

Depending on the approach, different key points were extracted:
Bounding boxes defining the rectangular regions around faces and hands.
Specific key points on the hands (e.g., fingertips, knuckles) crucial for sign language recognition.
Hand orientation relative to the camera.

4. An LSTM Deep Learning Model Was Built and Trained

Model Architecture: The model was designed in TensorFlow using layers like:
Convolutional Neural Networks (CNNs) to process the extracted key points or features, learning patterns from the data.
Long Short-Term Memory (LSTMs) to handle the sequence of key points from video frames, capturing temporal relationships within signs.
An output layer with a number of units equal to the number of signs to be recognized, classifying the input sequence and predicting the most likely sign.

5. Sign Language Predictions Were Made

Once trained, the model was used to predict signs from new video data:
Key points were extracted from the new video frames.
The sequence of key points was fed into the trained model.
The model predicted the most likely sign based on the learned patterns.

6. Model Weights Were Saved

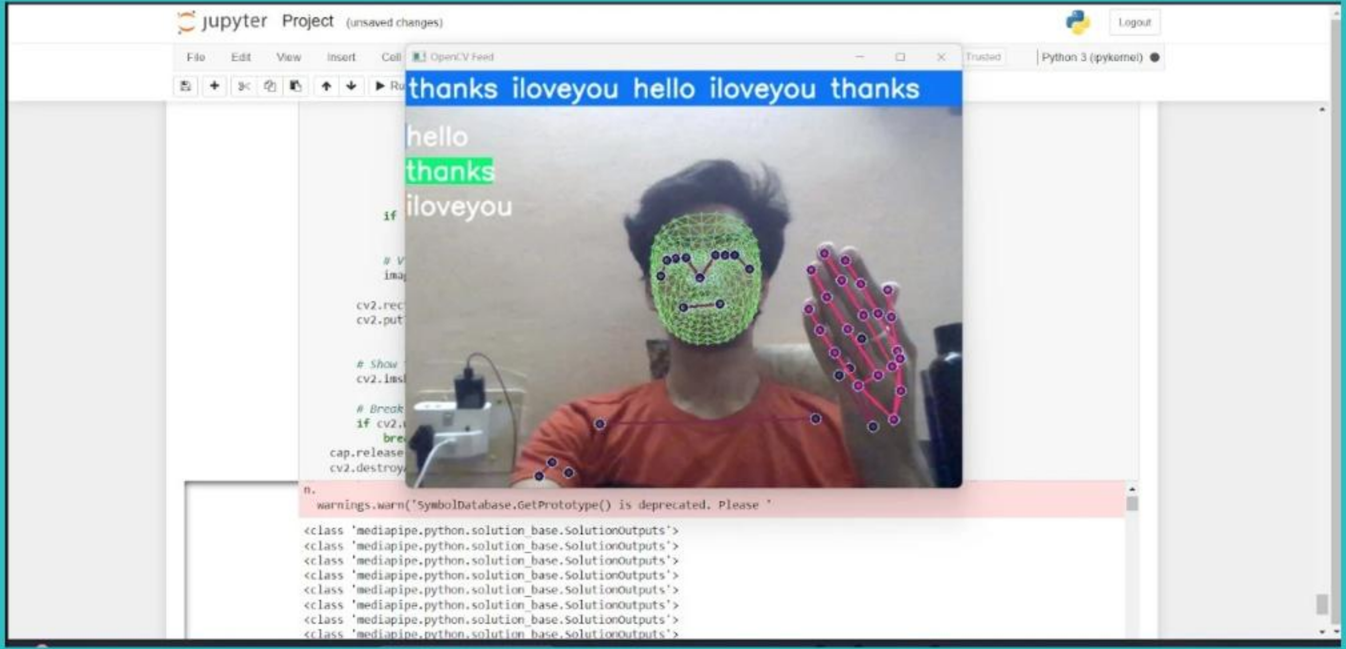
After training, the model's weights were saved for future use, allowing the trained model to be loaded without retraining every time. TensorFlow provides methods for saving and loading models.

7.Evaluation using a Confusion Matrix was conducted

A confusion matrix was used to evaluate the model's performance on the testing set. It visualized how often the model correctly predicted each sign and how often it made mistakes, helping identify areas for improvement.

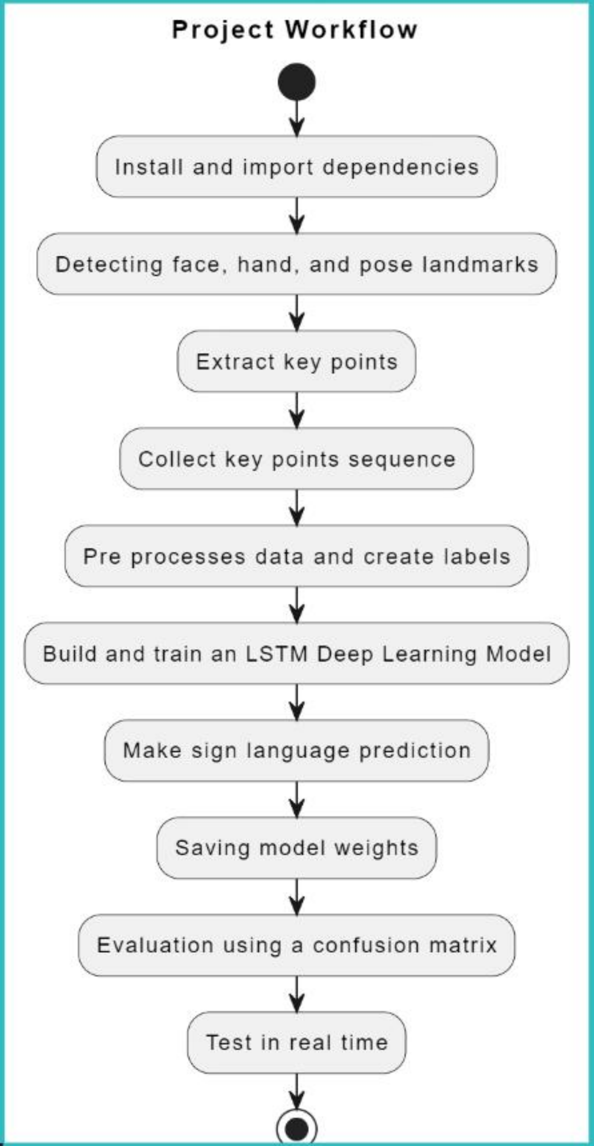
8. Real-Time Testing Was Performed

This step involved integrating the model into a real-time application:
Video frames were captured from a webcam or video source.
The frames were preprocessed, and key points were extracted.
The key point sequence was fed into the loaded model.
The predicted sign (text or spoken language) was displayed on the screen.
This required additional libraries for video processing and user interface creation.

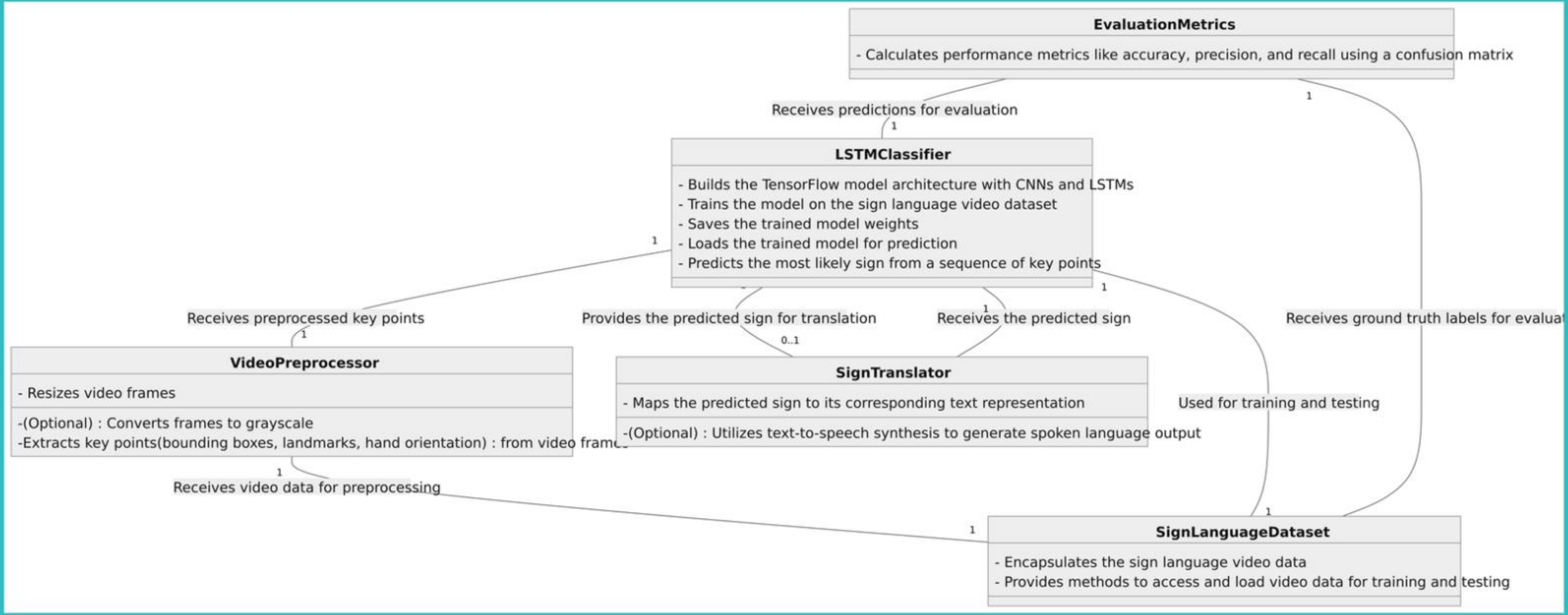

jupyter Project (unsaved changes)
Logout
File Edit View Insert Cell
OpenCV Feed
Trusted
Python 3 (ipykernel)
thanks iloveyou hello iloveyou thanks
hello
thanks
iloveyou

```
n.
  warnings.warn('SymbolDatabase.GetPrototype() is deprecated. Please '
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
```

Project Workflow

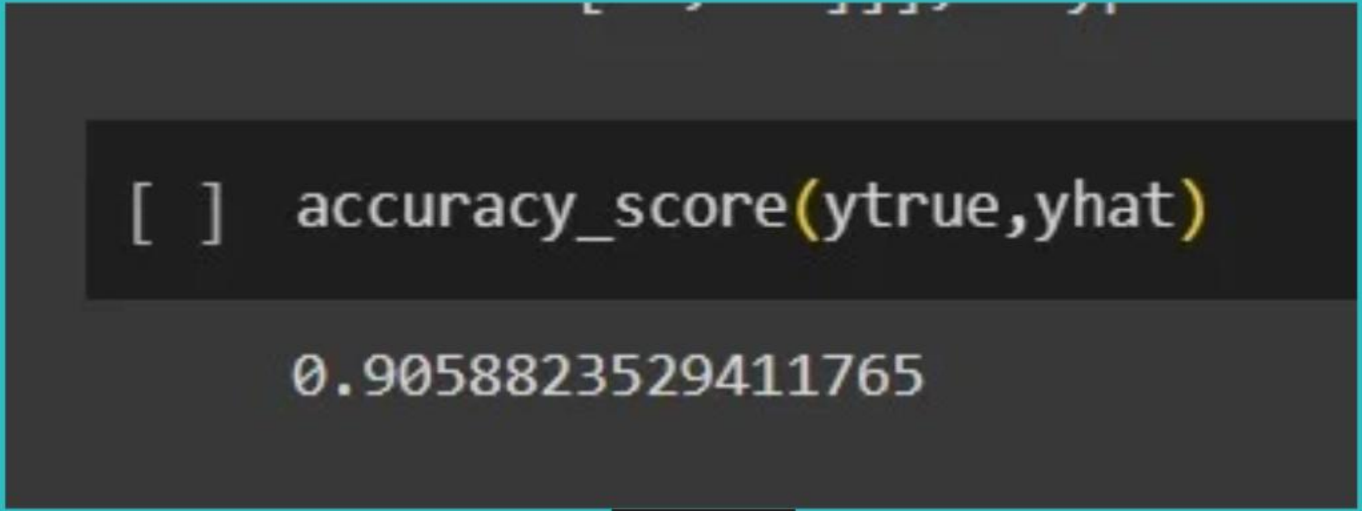

Project Workflow
Install and import dependencies
Detecting face, hand, and pose landmarks
Extract key points
Collect key points sequence
Pre processes data and create labels
Build and train an LSTM Deep Learning Model
Make sign language prediction
Saving model weights
Evaluation using a confusion matrix
Test in real time

UML Diagram


EvaluationMetrics
- Calculates performance metrics like accuracy, precision, and recall using a confusion matrix
Receives predictions for evaluation
1
1
LSTMClassifier
- Builds the TensorFlow model architecture with CNNs and LSTMs
- Trains the model on the sign language video dataset
- Saves the trained model weights
- Loads the trained model for prediction
- Predicts the most likely sign from a sequence of key points
1
1
Receives preprocessed key points
1
Provides the predicted sign for translation
0..1
Receives the predicted sign
1
Receives ground truth labels for evaluat
VideoPreprocessor
- Resizes video frames
-(Optional) : Converts frames to grayscale
-Extracts key points(bounding boxes, landmarks, hand orientation) : from video frame
SignTranslator
- Maps the predicted sign to its corresponding text representation
-(Optional) : Utilizes text-to-speech synthesis to generate spoken language output
Used for training and testing
1
Receives video data for preprocessing
1
1
SignLanguageDataset
- Encapsulates the sign language video data
- Provides methods to access and load video data for training and testing

Accuracy Score Calculation:

We calculated the accuracy score of the model on the testing set to quantify its overall performance. A high accuracy score indicates that the model is effectively interpreting sign language gestures.



```
[ ]  accuracy_score(ytrue,yhat)

     0.9058823529411765
```

Conclusion

In concluding our project on the Sign Language Interpretation System, we have achieved significant milestones in both technical implementation and performance. Through meticulous planning, rigorous testing, and adherence to industry standards, we have developed a solution that demonstrates robustness, scalability, and efficiency in interpreting sign language gestures.

Future Scope

1.Improved Accuracy and Performance
2. Real-time Applications
3. Multi-modal Fusion
4. Transfer Learning and Few-shot Learning
5. User Experience and Accessibility
6. Data Augmentation and Synthesis
7. Cross-modal Translation
8. Privacy and Ethical Considerations

Thank You!