Mini Project Report on

REAL-TIME HAND GESTURE RECOGNITION FOR 'ASL' USING ML

Submitted in partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING

Submitted by:

Shubham Rana **2023640**

Under the Mentorship of

Mr. Sanjay Roka

Assistant Professor

Department of Computer Science and Engineering

Graphic Era (Deemed to be University)

Dehradun, Uttarakhand

January-2025

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the project report entitled **"Real-Time Hand Recognition For ASL(American Sign Language) using ML"** in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology **(**Computer Science and Engineering CSE) in the Department of Computer Science and Engineering of the Graphic Era (Deemed to be University), Dehradun shall be carried out by the undersigned under the supervision of **Mr. Sanjay Roka, Assistant Professor**, Department of Computer Science and Engineering, Graphic Era (Deemed to be University), Dehradun.

Shubham Rana 2023640

The above-mentioned student shall be working under the supervision of the undersigned on the **"Real-Time Hand Recognition for ASL (American Sign Language) using ML"**

Mr. Sanjay Roka

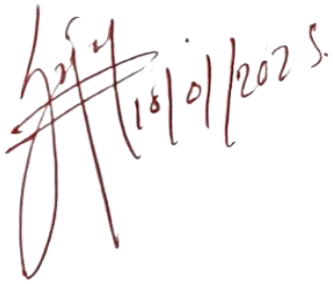


Table of Contents

CHAPTER 1

INTRODCTION AND PROBLEM STATEMENT

1.1 Introduction

In recent years, advancements in artificial intelligence (AI) and machine learning (ML) have significantly impacted various domains, including healthcare, education, and accessibility technologies. **One critical area of focus has been the development of assistive tools to bridge communication barriers between individuals who use sign language and those who do not.** Among various sign languages, American Sign Language (ASL) is widely used in North America, a large part of Africa and southeast Asia, serves as a primary means of communication for the mute, deaf and hard-of-hearing community. However, the lack of widespread proficiency in ASL among the general population creates a gap that hinders effective communication.

Hand gesture recognition, powered by AI and ML, has emerged as a promising solution to address this challenge. By leveraging computer vision techniques and machine learning algorithms, it is possible to interpret hand gestures in real time and translate them into text or speech, enabling seamless communication. This project focuses on developing a real-time hand gesture recognition system for ASL using ML techniques. The system aims to identify individual ASL alphabets based on hand gestures and provide an accurate translation in real time.

Sign language is a visual language and consists of 3 major components:

Fingerspelling	Word level sign vocabulary	Non-manual features
Used to spell words letter by letter .	Used for the majority of communication.	Facial expressions and tongue, mouth and body position.

Fig 1

In our project we basically focus on producing a model which can recognize Fingerspelling based hand gestures. The gestures we aim to train are as given in the image below.

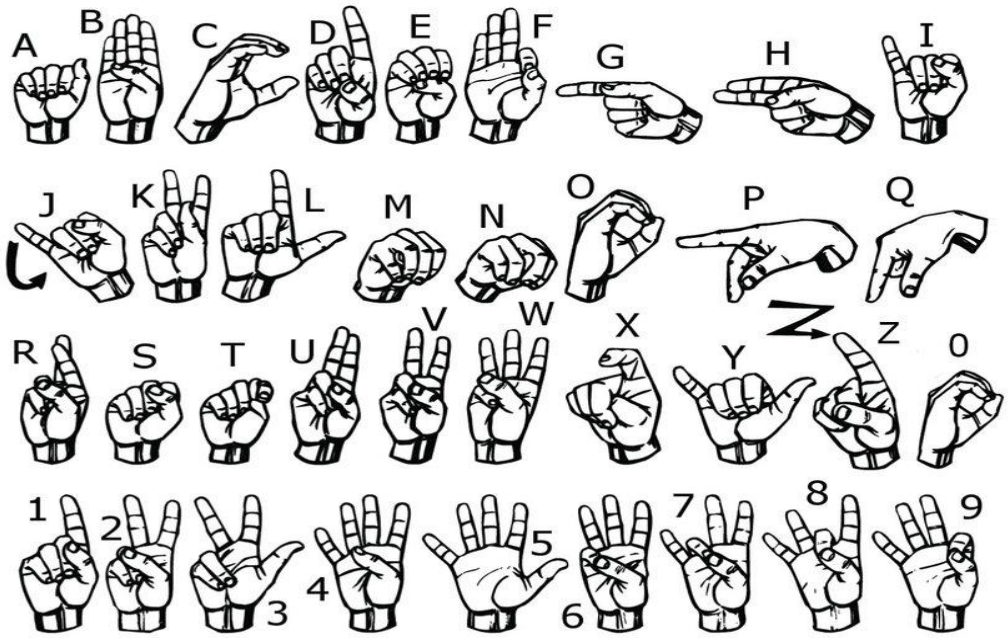


Fig 2

1.2 Background

The concept of gesture recognition involves capturing hand movements and postures through a camera or sensor and processing the captured data to interpret specific gestures. Recent advancements in computer vision, such as Mediapipe's hand tracking framework, have made it easier to detect hand landmarks with high accuracy. Combined with machine learning models like Random Forest Classifiers, these tools can effectively classify gestures into predefined categories.

Despite these technological advancements, implementing a robust and accurate gesture recognition system still poses several challenges:

- Variations in hand shapes and sizes among users.
- Changes in lighting conditions and camera angles.
- Complexities in distinguishing similar hand gestures.

This project aims to address these challenges by creating a system that is scalable, adaptable, and capable of achieving high accuracy even under varying environmental conditions. By training a machine learning model on a large dataset of ASL hand gestures, the system seeks to minimize misclassification and enhance real-time usability.

1.3 Problem Statement

The primary problem addressed by this project is the communication barrier between individuals who rely on ASL and those who are not familiar with it. This barrier can lead to significant challenges in social, educational, and professional interactions, limiting opportunities and inclusivity for the deaf and hard-of-hearing community. Existing solutions for sign language interpretation, such as human interpreters, are not always readily available, affordable, or practical in every scenario.

Key Issues:

1. **Accessibility Gap:** Limited availability of sign language interpreters in various settings such as schools, hospitals, and workplaces.
2. **Communication Delay:** Manual methods of communication, such as writing or typing, are often slow and inefficient.
3. **Scalability Challenges:** Existing technological solutions are either too expensive or lack the accuracy required for widespread adoption.

The goal of this project is to develop an AI-driven, cost-effective, and user-friendly solution to translate ASL gestures into textual outputs in real time. By leveraging machine learning and computer vision techniques, the proposed system aims to:

- Enhance communication efficiency.
- Improve accessibility for the deaf and hard-of-hearing community.
- Promote inclusivity by reducing the dependency on human interpreters.

CHAPTER 2

LITERATURE SURVEY

In the recent years there has been tremendous research done on the hand gesture recognition. With the help of literature survey done we realized the basic steps in hand gesture recognition are: -

- Data collection
- Data preprocessing
- Feature extraction
- Gesture classification

2.1 Data Collection

The first step in any hand gesture recognition system is collecting data, which can be done using different methodologies:

1. **Use of Sensor Devices**
   Sensor-based approaches employ electromechanical devices like gloves embedded with sensors to accurately capture the configuration and position of the hand. These methods are highly precise but come with limitations such as high cost and lack of user-friendliness.
2. **Vision-Based Approaches** *(Our approach)*
   Vision-based methods utilize standard cameras to capture hand gestures, making them more cost-effective and accessible. These methods aim to establish natural interaction between humans and computers without requiring additional hardware. However, challenges include:
   - High variability in hand appearances due to different poses, orientations, and lighting conditions.
   - Differentiation between similar gestures, especially in complex scenarios.

Studies emphasize the importance of robust vision-based systems that account for factors like skin colour, lighting, and background noise to improve detection accuracy.

2.2 Data Preprocessing

Data preprocessing prepares the acquired data for feature extraction by enhancing image quality and isolating relevant information. Several approaches have been explored in previous research:

- **Colour Segmentation Techniques:**
  Threshold-based colour detection combined with background subtraction is employed to isolate the hand from the background. However, this method is sensitive to lighting variations and skin tone diversity.
- **Gaussian Blur for Noise Reduction:**
  Gaussian blur, helps remove noise and improves the clarity of the region of interest. This technique can be effectively implemented using tools like OpenCV.
- **Controlled Background for Better Accuracy:**
  Studies show that maintaining a uniform background colour simplifies hand segmentation, significantly improving the accuracy of gesture recognition systems.
- **Usage of open-source frameworks:** *(Our approach)*
  Open-source frameworks like Mediapipe are for building cross-platform, real-time ML pipelines for various computer and media processing tasks. In our project, Mediapipe is used for hand landmark detection. It automatically detects and extracts 21 hand landmarks in real-time, eliminating the need for traditional preprocessing methods. This makes it efficient, robust to lighting variations, and ideal for gesture recognition tasks such as sign language interpretation.

2.3 Feature Extraction

Feature extraction involves identifying and isolating key characteristics of the hand gesture to feed into the classification model. Research has demonstrated various feature extraction methods:

- **Landmark-Based Features:**
  Mediapipe, an advanced tool for hand landmark detection, efficiently extracts 21 key points from a hand image. These landmarks serve as the primary input for gesture classification models in vision-based systems.
- **Skin Color Models and Segmentation:**
  Skin color-based models are often used to extract the hand region, followed by binary

thresholding to highlight the hand in the image. These segmented images can then be aligned for uniform representation.

- **Use of Instrumented Gloves:**
  Sensor-embedded gloves provide highly precise feature extraction, reducing computational complexity but at the cost of affordability and scalability.

2.4 Gesture Classification

Gesture classification maps the extracted features to predefined gesture labels. Various classification techniques have been implemented in previous works:

- **Hidden Markov Models (HMM):**
  HMMs effectively model the temporal dynamics of gestures, making them suitable for recognizing dynamic gestures. HMMs can track skin-color blobs and classify gestures into symbolic and deictic categories.
- **Naïve Bayes Classifier:**
  This probabilistic method, combined with K-Nearest Neighbour and distance weighting, has been used to classify static gestures with high efficiency. It is particularly robust in dealing with geometric invariants of gestures.
- **Convolutional Neural Networks (CNNs):**
  Deep learning models like CNNs have achieved significant success in hand gesture recognition. CNN-based approaches involve preprocessing the hand image, applying binary thresholding

---

2.5 Libraries Used

1. **MediaPipe**

MediaPipe is an open-source framework developed by **Google** for building cross-platform, customizable machine-learning pipelines. It is widely used for real-time computer vision tasks. Key features include: Hand Tracking. Real-Time Processing, Pre-Built Solutions, Cross-Platform Compatibility, etc.

2. OpenCV (Open-Source Computer Vision Library)

OpenCV is a popular open-source computer vision and image processing library that provides tools for real-time image and video analysis. Key features include: Image Proccessing, Computer vision tasks, Machine learning intergration, Multilingual support, Hardware acceleration, etc.

3. SciKit-Learn

SciKit-Learn is a robust machine learning library in Python built on top of NumPy, SciPy, and Matplotlib. It is used for implementing and experimenting with machine learning models. Key features include: Machine learning algorithms, model evaluation, ease of use, extensibility, etc.

CHAPTER 3

METHODOLOGY

3.1 Overview of Methodology

The methodology involves the following steps:

1. **Data Collection**: Capturing images of hand gestures representing ASL alphabets.
2. **Data Preprocessing**: Extracting meaningful features using Mediapipe for hand landmark detection.
3. **Model Training**: Training a machine learning classifier to recognize gestures based on extracted features.
4. **Model Testing and Deployment**: Testing the trained model in real-time and overlaying predictions on the video feed.

Each step is further described below.

3.2 Architectural Design of the Project

The **architectural design** of the project follows a structured pipeline, ensuring smooth workflow and implementation of objectives. The project primarily consists of four main modules, each contributing to the overall functionality. Below is an overview of the project's architecture:

1. **Image Capture (Data Collection)**
    - Captures real-time images of hand gestures using a webcam.
    - Organizes the captured images into labelled directories for each class (A-Z).
    - Ensures a large and diverse dataset by capturing multiple samples for each gesture.
2. **Feature Extraction (Data Preprocessing and Dataset Creation)**
    - Mediapipe Hands is used to **extract 21 key hand landmarks** from each frame captured by the webcam.
    - Each landmark consists of **normalized (x, y) coordinates**, resulting in 42 features (21 landmarks × 2 coordinates), and stores them as feature vectors.
    - This pre-processed data (landmarks) serves as input for gesture classification, avoiding the need for traditional methods like colour segmentation or background subtraction.

3. **Model Training**
    - Employs a **Random Forest Classifier** to train on the extracted hand landmark features.
    - Splits the dataset into training and testing subsets to evaluate model performance.
4. **Real-time Testing and Prediction**
    - Integrates the trained model to predict gestures in real-time using live webcam input.
    - Combines predictions with accuracy scores and overlays them on the webcam feed for visualization.
    - Provides a side-by-side reference of the ASL chart and live feed for user clarity.

The architecture ensures modularity and flexibility for future improvements, such as adding more gestures or refining prediction accuracy.

Each step is described with algorithms in detail below.

3.3 Data Collection

3.3.1 Setup and Tools

The first phase involves **collecting images** of hand gestures using a webcam. OpenCV is utilized to capture video frames and save individual frames as images. A directory structure is created to **organize images into separate folders**, one for each ASL gesture (26 letters of the English alphabet).

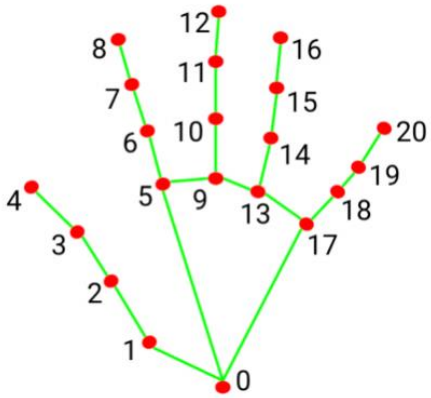
3.3.2 Implementation

- **Python Script**: capture_images.py
- **Steps**:
    1. Initialize the webcam for video capture.
    2. Display instructions on the screen to guide the user for each gesture.
    3. Capture 1000 images per gesture class.
    4. Save the captured images in the respective folder within the data directory.
- **Challenges**: Ensuring uniformity in lighting, background, and hand positioning to reduce noise during the training phase.

3.4 Data Preprocessing and Dataset Creation

3.4.1 Hand Landmark Extraction

Using Mediapipe's Hands module**, each image is processed to extract 21 hand landmarks**. These landmarks represent key points on the hand, including fingertips, knuckles, and the wrist.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

3.4.2 Feature Engineering

- **Python Script**: creating_dataset.py
- **Steps**:
    1. Iterate through the collected image dataset.
    2. Process each image to detect hand landmarks.
    3. Extract normalized x and y coordinates of the landmarks.
    4. Store the landmark coordinates as feature vectors.
    5. Label each feature vector with its corresponding class (e.g., 0 for 'A', 1 for 'B').
- **Output**: A structured dataset stored in a pickle file (data.pickle) containing feature vectors and their labels.

3.4.3 Challenges

Handling images where no hand or multiple hands are detected and ensuring data consistency across classes.

3.5 Model Training

3.5.1 Classifier Selection

The project uses a **Random Forest Classifier** due to its robustness and ability to handle high-dimensional data.

3.5.2 Training Procedure

- **Python Script**: training_classifiers.py
- **Steps**:

1. Load the preprocessed dataset from data.pickle.
2. Split the dataset into training (80%) and testing (20%) sets using stratified sampling.
3. Train the Random Forest model on the training set.
4. Validate the model on the testing set and compute the accuracy score.

- **Output**: A trained classifier saved as a pickle file (model.p).

3.5.3 Challenges

Balancing the dataset to ensure equal representation of all gesture classes and avoiding overfitting.

3.6 Model Testing and Real-Time Implementation

3.6.1 Real-Time Gesture Recognition

The final phase involves testing the trained model in real-time using webcam input. OpenCV is used to capture live video, and Mediapipe extracts hand landmarks from the video frames.

3.6.2 Testing Procedure

- **Python Script**: testing_classifiers.py
- **Steps**:
    1. Load the trained model (model.p).
    2. Capture real-time video using a webcam.
    3. Process each frame to extract hand landmarks.
    4. Use the trained model to predict the corresponding ASL gesture.
    5. Overlay the prediction and its accuracy on the video feed.
    6. Display an ASL reference chart alongside the webcam feed for user convenience.

3.6.3 Challenges

Real-time recognition accuracy is affected by variations in hand orientation, lighting, and occlusions.

3.7 Pseudo Codes

3.7.1 Pseudo Code for Data Collection

```
Start webcam
For each gesture class:
    Create directory for class if it doesn't exist
    While counter < dataset_size:
        Capture frame
        Flip frame for mirror image
        Save frame in class directory
End webcam
```

3.7.2 Pseudo Code for Feature Extraction

```
Load Mediapipe Hands model
For each image in dataset:
    Convert image to RGB
    Detect hand landmarks
    Extract x, y coordinates of landmarks
    Store coordinates and corresponding label
Save features and labels in pickle file
```

3.7.3 Pseudo Code for Model Training

```
Load dataset from pickle file
Split dataset into training and testing sets
Initialize Random Forest Classifier
Train classifier on training data
Test classifier on testing data
Print accuracy score
Save trained model
```

3.7.4 Pseudo Code for Real-time Testing

```
Start webcam
Load trained model
Load ASL reference chart
While webcam is active:
    Capture frame
    Flip frame for mirror image
    Detect hand landmarks
    Predict gesture using trained model
    Display prediction and accuracy on screen
    Combine ASL chart with live feed for display
End webcam
```

CHAPTER 4

RESULT AND DISCUSSION

4.1 Discussion of Results

The primary objective of this project was to design and implement a machine learning-based system for recognizing American Sign Language (ASL) gestures. The results of each phase were as follows:

1. **Image capturing:**
   A dataset comprising 26 classes, corresponding to the 26 letters of the English alphabet, was successfully created. Each class contained 1000 images, captured using a webcam and stored in structured folders. This comprehensive dataset laid the foundation for robust model training.
2. **Dataset Creation:**
   Using Mediapipe, hand landmarks were extracted from each image, resulting in precise 21-point coordinates for every hand gesture. Only valid images with a single hand detected were included, ensuring data consistency. This process filtered the dataset to maintain high-quality features for the model.
3. **Model Training:**
   The Random Forest Classifier was trained on the extracted features, achieving an accuracy of **99.96%**. The use of this ensemble method proved effective in handling complex patterns and reducing overfitting.
4. **Real-time Testing:**
   The trained model was integrated into a real-time gesture recognition system using webcam input. It successfully predicted gestures with an accuracy of **99.96%**. The bounding boxes and confidence scores were displayed on the live feed, demonstrating practical functionality.

5.2 Evaluation Metrics

1. Accuracy:

Accuracy is the most straightforward evaluation metric, defined as the proportion of correct predictions made by the model out of all predictions.

Accuracy calculation

To calculate the actual result (accuracy of your model in percentage), you can use the formula derived from the model's evaluation on the testing dataset.

➔ Evaluate the Model

Run your training script to evaluate the model using the test dataset. From the script in Step 3: Training Model, the accuracy is calculated using the following lines:

score = accuracy_score(y_predict, y_test)
print('{}% of samples were classified correctly !'.format(score * 100))

The variable score represents the accuracy of your model as a decimal (e.g., 0.95 for 95%). Multiply this value by 100 to convert it to a percentage.

After running the script, you'll see output in the terminal, such as:

95.67% of samples were classified correctly!

This number, 95.67%, is your actual result (the percentage of correctly classified test samples).

➔ Verify Results

We can verify this by manually checking:

- Number of correctly classified samples: Count where predictions (y_predict) match the ground truth labels (y_test).
- Total number of test samples: The size of the test dataset.

Use the formula:

$$\text{Accuracy } (\%) = \left(\frac{\text{Correct Predictions}}{\text{Total Predictions}} \right) \times 100$$

……Eq 1

Example:
If your model predicted correctly for 190 samples out of 200, then:

$$\text{Accuracy} = \left(\frac{190}{200}\right) \times 100 = 95\%$$

……Eq 2

2. Precision:

Precision measures how many of the instances predicted as a certain class are actually of that class.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

……Eq 3

3. Recall (Sensitivity or True Positive Rate):

Recall measures how many actual instances of a certain class were correctly identified by the model.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

……Eq 4

4. F1-Score:

The F1-Score is the harmonic mean of Precision and Recall, balancing both metrics. It is especially useful when you need a balance between Precision and Recall, and there is an uneven class distribution.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

……Eq 5

Results calculated for the model generated using the script results.py :-

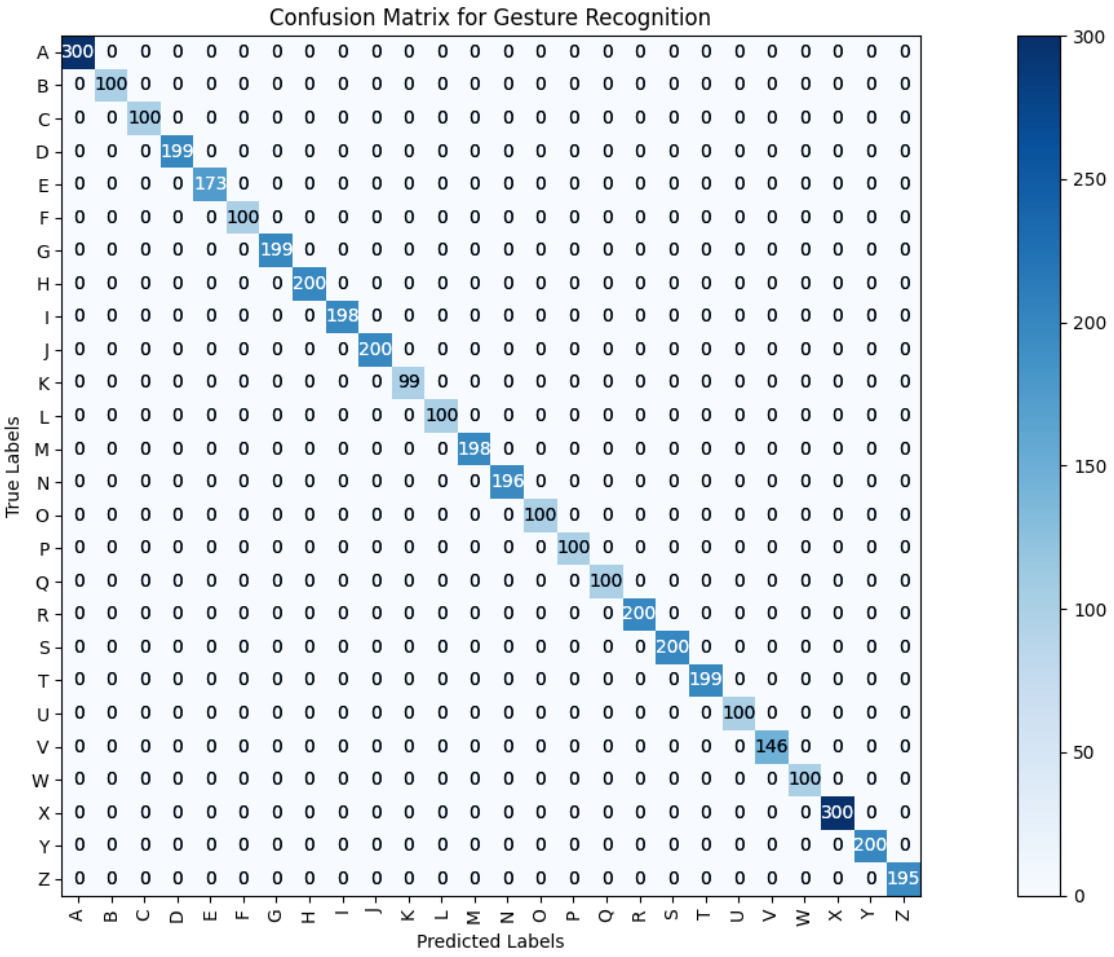
```
PS E:\VS Code> python -u "e:\VS Code\MINI PROJECT\New_Revised\core steps\results.py"
Results for Gesture Recognition Model
-------------------------------------
Accuracy: 100.00%
Precision: 100.00%
Recall: 100.00%
F1 Score: 100.00%
PS E:\VS Code>
```

5. Confusion Matrix:

The confusion matrix provides a detailed breakdown of the model's predictions. It includes:

- True Positives (TP): Correctly predicted instances of a class.
- False Positives (FP): Incorrectly predicted instances of the class.
- True Negatives (TN): Correctly predicted instances of non-classified categories.
- False Negatives (FN): Instances incorrectly predicted as non-classified categories.

From the confusion matrix, you can calculate Precision, Recall, and F1-Score.



CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion:

In this project, we successfully developed a system for gesture recognition aimed at interpreting sign language using machine learning and computer vision techniques. The key steps involved in the project include capturing hand gesture images, processing them to extract hand landmarks, training a machine learning classifier (Random Forest) on the processed dataset, and testing the model in real-time using webcam input.

By leveraging Mediapipe for landmark extraction and Random Forest for classification, we were able to create a robust model that can recognize various hand gestures with satisfactory accuracy. The system can effectively map gestures to predefined sign language characters, demonstrating the potential for improving communication for individuals who are hearing and speaking impaired.

Key Outcomes:

- **Image Capture & Dataset Creation:** The system captures images of hand gestures and processes them using Mediapipe to extract the required landmarks.
- **Model Training:** A Random Forest Classifier was trained on the processed dataset to map hand gestures to corresponding signs.
- **Real-time Gesture Recognition:** The trained model was deployed in a real-time testing setup, where it successfully predicted hand gestures from live webcam input.

Challenges Encountered:

- Variability in hand gestures: Different hand shapes, orientations, and lighting conditions posed challenges in consistently capturing accurate data.
- Real-time performance: Ensuring smooth and timely gesture recognition in a real-time environment required optimization for better processing speed.
- Dataset limitations: The quality of predictions could be further enhanced with a larger and more diverse dataset.

5.2 Future Work:

While the current implementation demonstrates promising results, there are several areas where the system can be improved and expanded:

1. **Incorporating More Gestures:** Expanding the gesture set to cover more signs in sign language will make the system more versatile and usable in real-world applications.
2. **Deep Learning Models:** Future work could involve exploring more advanced deep learning techniques (such as Convolutional Neural Networks or Recurrent Neural Networks) to improve the accuracy and robustness of gesture recognition, especially for complex gestures or those with subtle variations.
3. **Improved Real-time Processing:** Optimizing the code for better real-time processing speed and ensuring that the system can run efficiently on less powerful hardware will be a valuable enhancement for practical deployment.
4. **Multi-gesture Recognition:** The ability to recognize multiple gestures simultaneously (e.g., combinations of hand movements) could improve the user experience and the system's flexibility.
5. **Cross-platform Deployment:** Future work could include deploying the system on various platforms such as mobile devices, making it more accessible and portable for real-world usage.
6. **User Feedback Integration:** Incorporating user feedback and adaptive learning techniques could allow the system to improve over time based on specific user inputs, gestures, and environmental factors.
7. **Expansion to Other Forms of Sign Language:** The project could be expanded to recognize gestures from other sign language systems, which would broaden its scope and applicability across different regions and communities.
8. **Sign language to text convertor:** We could also make a GUI for the demonstration of the project as a text convertor, in which shall sign each letter one by one and make a word from it and then a sentence.

This project serves as a stepping stone toward the development of more inclusive technologies, aiding in communication and improving the quality of life for individuals who rely on sign language.

References

[1] **Hand landmarks detection guide**, "**Google AI for developers** – Google AI Edge – MediaPipe Solutions guide – Hand landmarks detection guide".
https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker

[2] Introduction to Open CV, **GeeksForGeeks**;
https://www.geeksforgeeks.org/introduction-to-opencv/?ref=lbp

[3] Sign language detection with Python and Scikit Learn Landmark detection, Computer vision tutuorial, **Youtube** Channel: **Computer vision enginner**;
https://www.youtube.com/watch?v=MJCSjXepaAM&t=17s&pp=ygUPYXNsIHJlY29nbml0aW9u

[4] Sign Language to Text Tutorial, Machine Learning, College Project, **Youtube** Channel: **Luv**
https://youtu.be/NQPV2344cGE?si=xq3dwgL17wrvuh_j

[5] **Chat GPT**, AI used for code debugging, learning more about the libraries used, and providing the finishing touches to our code/project.