AI CLINIC PROJECT

Sign Numbers

BY NOUR YAAKOUB
EYA BEN ISMAIL
IKRAM KALKOUL

Overview

AI CLINIC PROJECT

01

Abstract

This project develops a system to recognize hand signs for numbers 0–9 using machine learning. Images are preprocessed and used to train and test models, with a menu-driven interface allowing algorithm selection, model evaluation, and real-time webcam predictions. The goal is to support sign language recognition for improved communication accessibility.

Data Collection

Method:

- Images were captured manually using a smartphone camera. (960 x 1280 px)
- Multiple individuals participated to add variation in hand shape and size.

Challenges:

- Ensuring consistency in hand position.
- Managing variations in lighting.

sample of the collected data

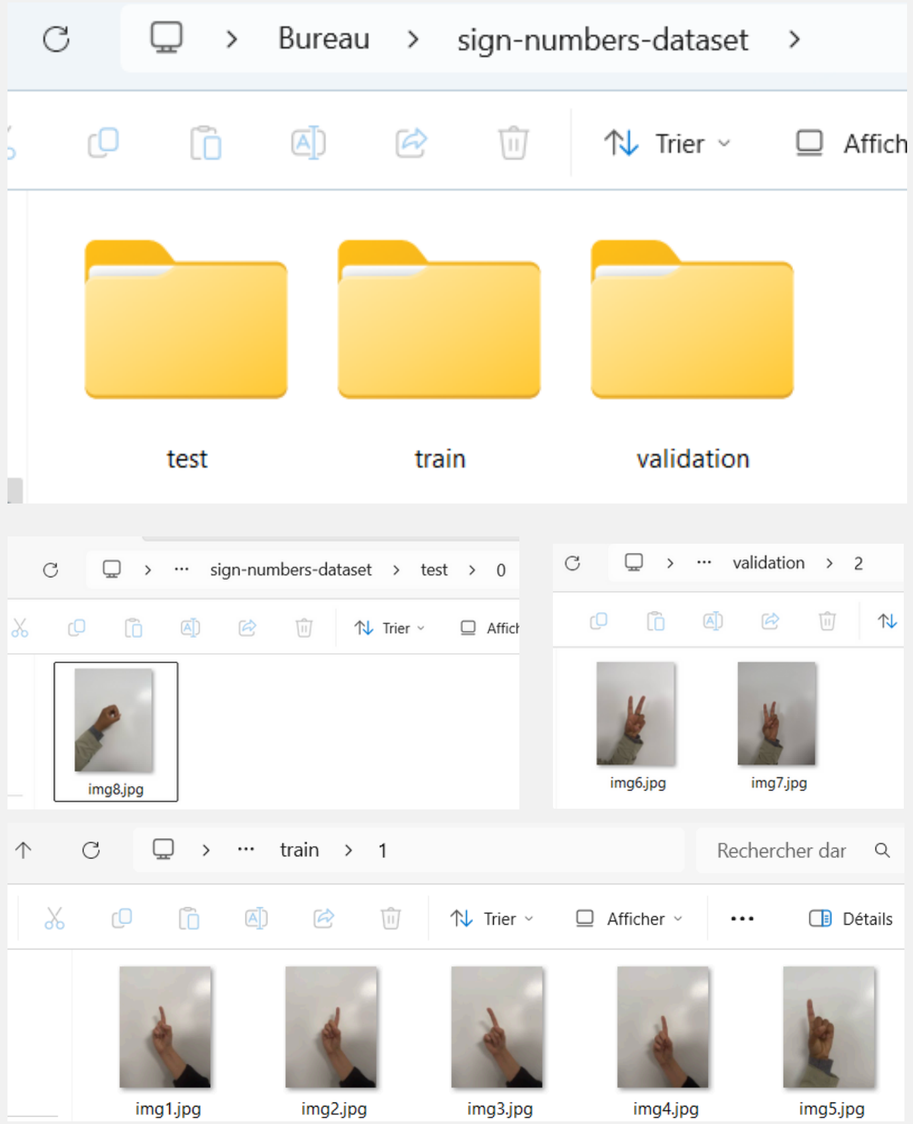
Data Collection

Dataset Structure:

- Organized into three folders: train, validation, and test.
- Each class (digit) has its own subfolder.

Total Images:

- 8 images per class .
- Balanced across all digits to avoid class bias.
- Train folder has 5 images, test folder has 1 image and validation folder has 2 images for each digit.



Data preprocessing

Objective:

To preprocess and augment hand gesture images for training machine learning models.

Key Features:

- Hand detection using a custom module (HandTrackingModule).
- Image augmentation to improve model robustness.
- Structured dataset creation for train, test, and validation sets.

Data preprocessing

AI CLINIC PROJECT

Hand Detection and Image Processing Sequence

Hand Detection

Detect hands using landmark tracking

Crop ROI

Crop Region of Interest around the hand

Skip Unreadable Images

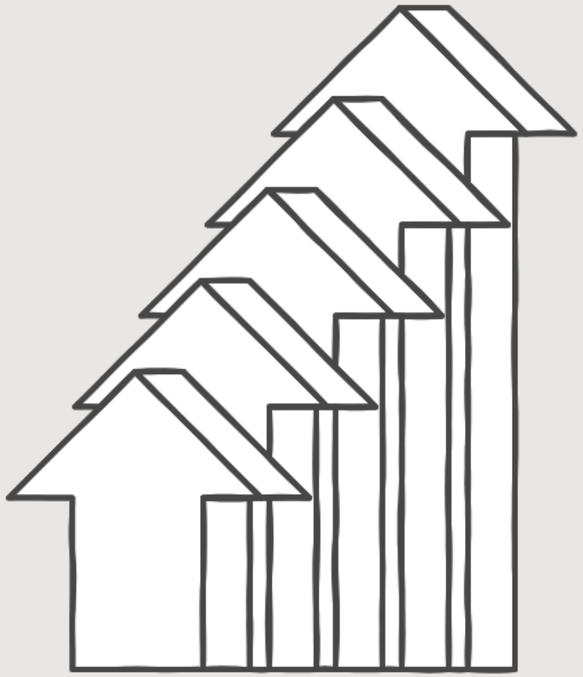
Skip images that are unreadable or have no hand detected

Apply Augmentation

Apply brightness changes, rotation, and flipping to the ROI

Save Augmented Images

Save augmented images with modified filenames



Cleaning Steps:

- Skipped unreadable or blank images.
- Ignored images where no hand was detected.

Data Augmentation:

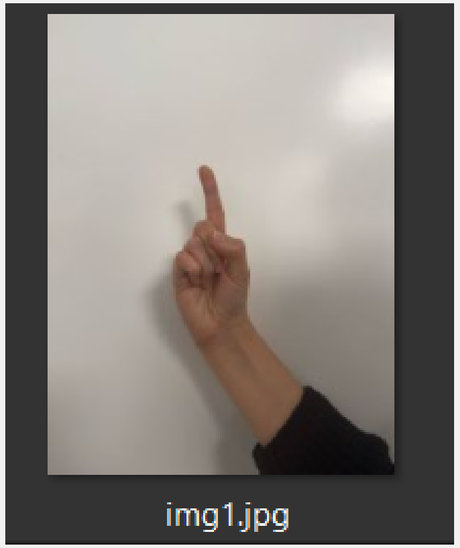
- Brightness adjustment (e.g., using different alpha values).
- Rotation (by specified angles).
- Flipping (horizontal, vertical, etc.).
- Augmented images were saved with modified names.
- For each class we went from 8 images to 136 images

Data preprocessing

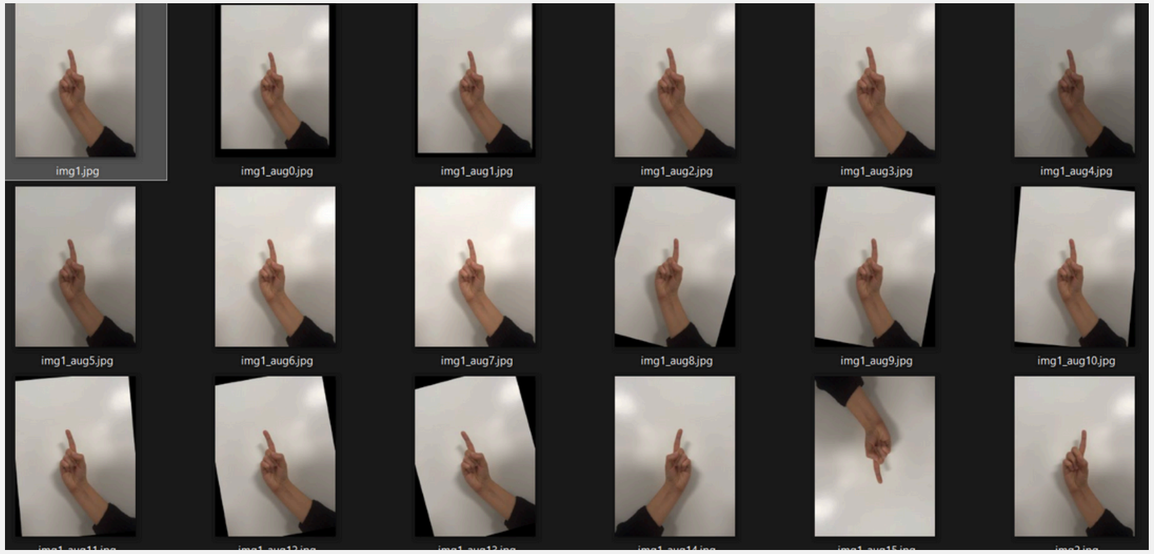
Result:

A clean, uniform, and enriched dataset prepared for model training.

from



to...



Data preprocessing

OG data

- 1 image for test.
- 5 images for training.
- 2 images for validation.
- 8 images in total per class

preprocessed data

- 17 image for test.
- 85 images for training.
- 34 images for validation.
- 136 images in total per class
- NOTE: for each images, we generate 16 more, the preprocessed folder contain the OG image + the new 16 images

Core components

The program follows a modular design pattern :

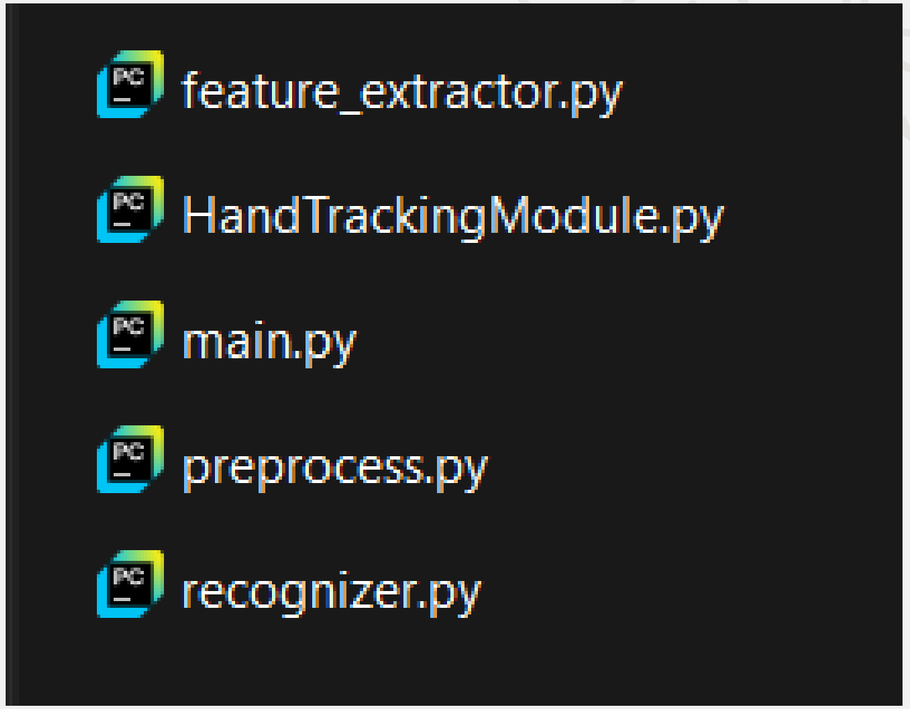
preprocess.py

main.py

HandTrackingModule.py → 1.Hand Tracking Module

feature_extractor.py → 2. Feature Extractor

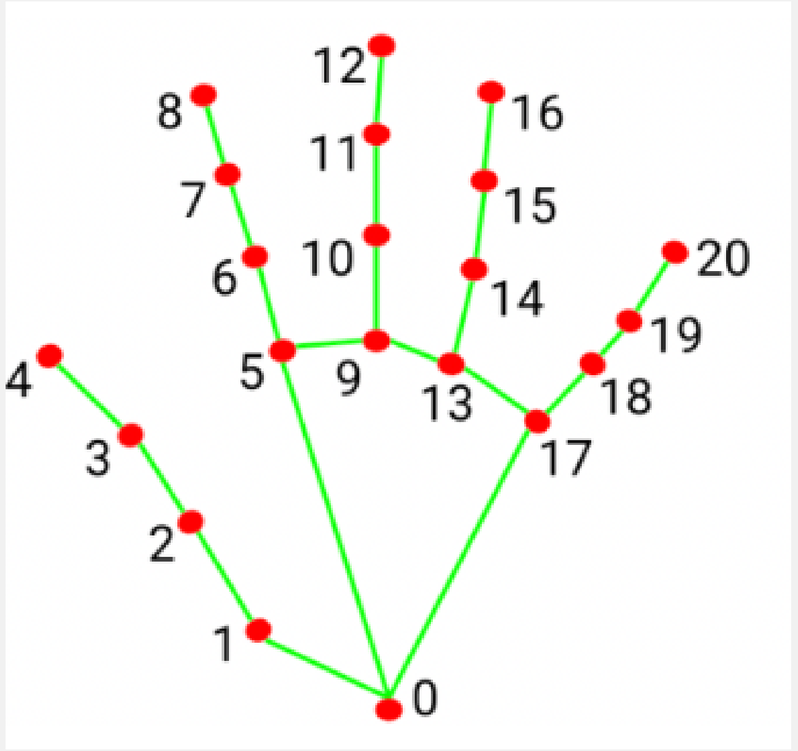
recognizer.py → 3. Number Sign Recognizer



Hand Tracking Module

This module uses MediaPipe, a powerful computer vision library, to detect and track hand landmarks in real-time video:

- Identifies 21 key points on the hand (joints and fingertips)
- Draws connections between these points to visualize the hand skeleton
- Provides essential position data for feature extraction

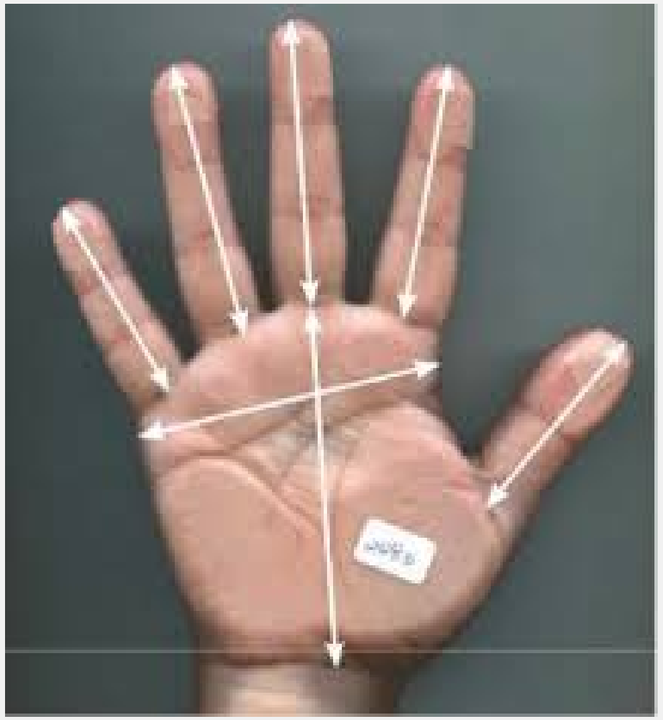


0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP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

Feature Extractor

The *HandFeatureExtractor* class extracts meaningful features from hand gestures:

- Normalizes hand landmarks to be invariant to position and scale
- Calculates 84 features total:
  - 42 raw features (x,y coordinates of the 21 landmarks)
  - 42 engineered features including:
    - Distances from palm to each fingertip
    - Distances between adjacent fingertips
    - Angles at each finger joint (15 angles)



Number Sign Recognizer

The main *NumberSignRecognizer* class manages the entire recognition process:

Dataset handling:

- Supports organized datasets with train/validation/test splits
- Processes images from each number's directory (0-9)
- Extracts and stores features for model training

```
def load_dataset(self, respect_splits=False):
    """
    Load images from dataset directories and extract features

    Args:
        respect_splits: If True, return separate train/val/test datasets
                        If False, combine all data into one dataset
    """
```

```
def _process_folder(self, folder_path, label, X, y):
    """Process all images in a folder"""
```

Number Sign Recognizer

Model training:

- Uses a Random Forest Classifier (Default configuration: 200 trees, max depth of 20)
- Can perform hyperparameter optimization with GridSearchCV (available for n_estimators, max_depth, min_samples_split, min_samples_leaf)
- Analyzes feature importance
- Saves trained models for future use

```
# Train with default parameters
print("Training model...")
self.model = RandomForestClassifier(
    n_estimators=200,
    max_depth=20,
    min_samples_split=2,
    min_samples_leaf=1,
    random_state=42
)
```

```
# Use grid search to find optimal hyperparameters
print("Optimizing model hyperparameters...")
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

Why random forest?

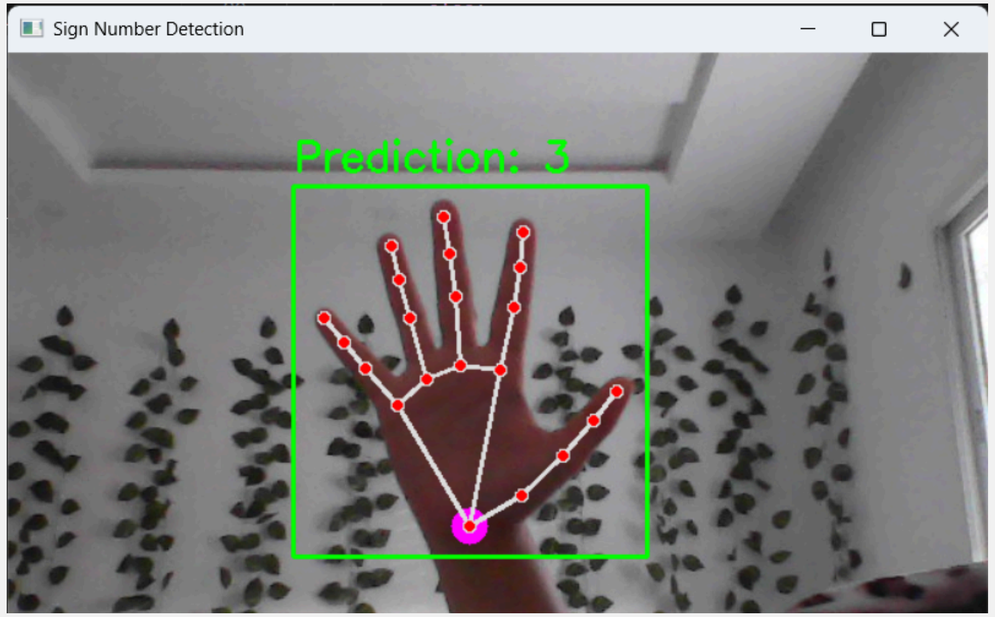
We actually tested our project with multiple algorithms. The issue we encountered that the model accuracy in those cases will be high, but when testing them with the webcam, the classification is wrong!

That is why we opted for machine learning algorithms, specifically Random Forest!

```
Select Model Type:
1. Simple CNN
2. Random Forest
3. K-Nearest Neighbors (KNN)
4. KMeans (Unsupervised)
```

```
Simple CNN Model created.
```

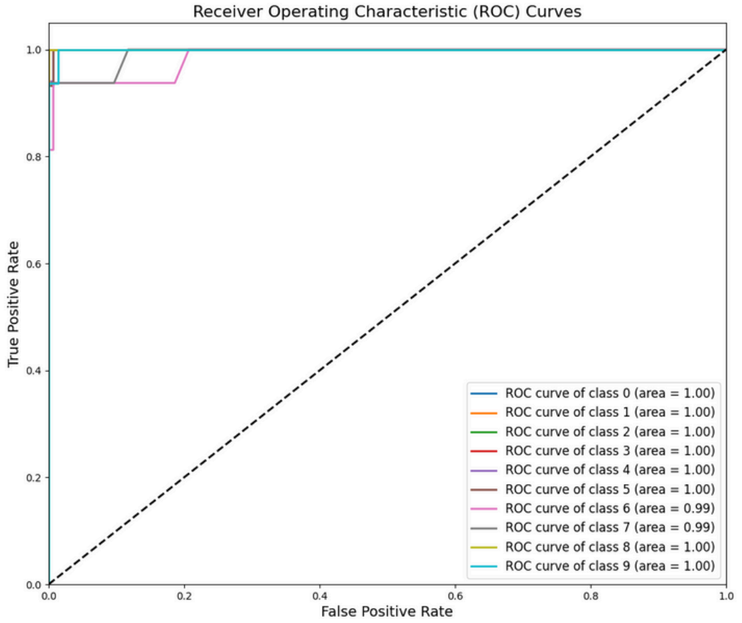
```
Epoch 10/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8908 - loss: 0.3673
```

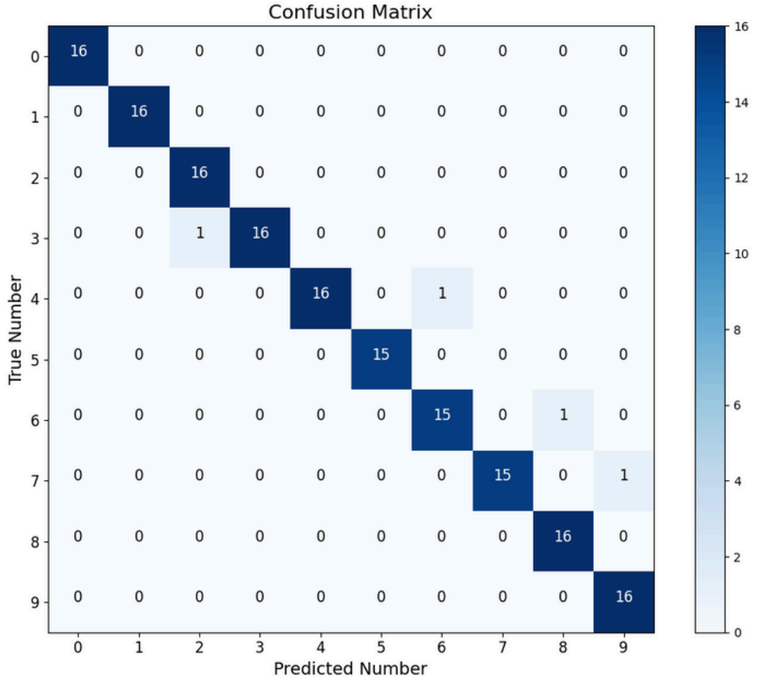


Number Sign Recognizer

evaluation metrics:

- ***IoU (Intersection over Union) Calculation***:
- Calculates IoU for each class by finding the ratio of true positives to the sum of true positives, false positives, and false negatives
- ***F1 Score*** :
- F1 score is the harmonic mean of precision and recall, giving a balanced measure of model performance
- ***ROC Curve*** :
- Creates ROC (Receiver Operating Characteristic) curves for each class using a one-vs-rest approach
- ***Confusion Matrix***

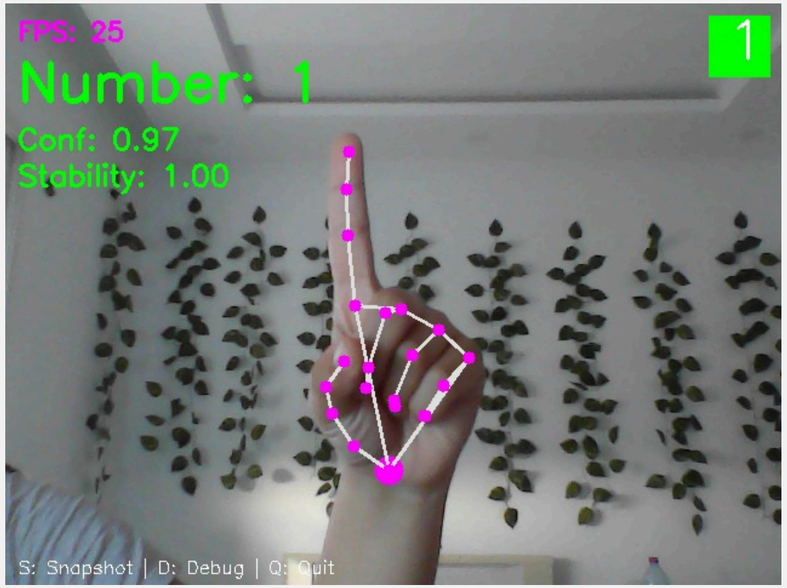




Number Sign Recognizer

Real-time detection:

- *Real-time visual feedback with color-coded confidence indicators (green, yellow and red)*
- *FPS counter to monitor performance*
- *Debug mode for detailed prediction information*



no debug mode

debug mode ON

05

Applications and Extensions

This system could be extended for:

- Educational tools for sign language learning
- Accessibility interfaces for communication
- Gesture-controlled applications
- Human-computer interaction research

The modular design makes it straightforward to integrate with other systems or expand to recognize additional gestures beyond numbers

Conclusion

This project represents not just a technical solution, but a commitment to creating technology that serves human values of connection, communication, and inclusion.

Thank you