WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

AI Sign Language Recognition

Dhariha V[1], Abinaya M[2], Dharshini Priya S[3], Aarthi V G[4], Dr. Kamalaveni V[5]

[1-4] *Student, Department of Artificial Intelligence and Data Science,* Sri Shakthi Institute of Engineering and Technology
[5]*Tutor, Department of Artificial Intelligence and Data Science,* Sri Shakthi Institute of Engineering and Technology
An Autonomous Institution, Affiliated to Anna University
Accredited by NAAC with "A" Grade COIMBATORE – 62

ABSTRACT—

Communication between sign language users and people who do not understand sign language remains a major social and accessibility barrier. Human interpreters are not always available, and most digital tools do not support sign language input. This project, titled AI-Based Sign Language Recognition and Text Translation, presents a web-based application that recognizes hand signs through a camera and converts them into readable text in real time. The system uses OpenCV for video capture, MediaPipe for hand detection, and a Convolutional Neural Network (CNN) built with TensorFlow/Keras for sign classification. The predicted text is delivered through a REST API to a React.js frontend, where it is displayed instantly to the user. By removing the need for a human interpreter in everyday situations, the proposed system improves accessibility and supports inclusive communication.

***Index Terms*—Sign Language Recognition, Indian Sign Language, Computer Vision, Deep Learning, CNN, MediaPipe, React.js, Accessibility**

I. INTRODUCTION

Sign language is the primary mode of communication for millions of people who are deaf or hard of hearing. It uses hand shapes, movements, and orientations to express letters, words, and ideas. However, most people do not understand sign language, which creates a communication gap in education, healthcare, workplaces, and public services.

Professional interpreters can bridge this gap, but they are costly, limited in number, and not available at all times. Advances in computer vision and deep learning now make it possible to build automated systems that recognize gestures from a simple camera feed and translate them into text.

The proposed project is an AI-powered application that captures hand gestures through a camera, processes the video frames using a trained model, identifies the corresponding sign, and displays the translated text on a web frontend. The system combines OpenCV and MediaPipe for hand detection, a CNN for classification, and React.js for an interactive user interface.

Despite its benefits, the system may face challenges such as variation in lighting, background clutter, and differences in signing style between users. These can be reduced through better datasets, data augmentation, and future improvements such as continuous sentence recognition and mobile support.

The main contributions of this work are:

- A camera-only sign recognition pipeline that needs no gloves or sensors.
- A CNN classifier trained for reliable recognition of a defined sign vocabulary.
- A REST-based design that connects the AI model with a modern React web interface.
- A modular and extensible architecture suitable for future sentence-level translation.

The rest of this paper is organized as follows. Sections II to V describe the problem, objectives, scope, and related work. Sections VI to XVI present the existing and proposed systems, architecture, modules, methodology, and implementation. Sections XVII onward cover testing, results, discussion, applications, and future directions.

II. PROBLEM STATEMENT

People who use sign language face daily difficulty in communicating with those who do not know it. Human interpreters are not always accessible, and written communication is slow and unnatural for many sign language users.

Existing digital tools mainly support spoken or written language and rarely accept sign language as input. Many existing recognition systems are also limited by requirements such as special gloves or sensors, controlled backgrounds, or high-end hardware, which makes them impractical for common use.

Furthermore, most research systems focus on American Sign Language, and there are fewer accessible tools for Indian Sign Language. Therefore, there is a clear need for a low-cost, camera-based system that recognizes signs in real time and presents the result as text through a simple web interface.

III. OBJECTIVES

- To develop an AI-based system that recognizes hand signs from a live camera feed and converts them into text.
- To detect and track hands accurately using OpenCV and MediaPipe.
- To train a CNN model using TensorFlow/Keras for reliable sign classification.
- To build a responsive React.js frontend that displays predictions in real time.
- To connect the frontend and the AI model through a REST API.
- To reduce the dependence on human interpreters in everyday communication.
- To design a scalable system that can later support more signs, sentences, and mobile devices.

IV. SCOPE OF THE PROJECT

The scope of this project covers a web-based application that recognizes a defined set of hand signs through a standard webcam and displays the translated text. The target users are sign language users and people who want to communicate with them.

The frontend is developed using React.js, HTML, CSS, and JavaScript. The recognition pipeline is implemented in Python using OpenCV, MediaPipe, and TensorFlow/Keras, and is exposed through a Flask-based REST API. An optional MongoDB database can store user history.

The initial version focuses on isolated sign recognition and does not include full sentence-level translation, text-to-sign conversion, or a dedicated mobile application. These are considered for future enhancement.

V. LITERATURE SURVEY

Early sign language recognition systems relied on sensor-based approaches such as data gloves and motion sensors. Although accurate, they are expensive and uncomfortable, which limits their real-world adoption.

Vision-based approaches later replaced sensors with cameras and used handcrafted features such as skin-color segmentation, contours, and histograms combined with classifiers like SVM and k-NN. These methods are sensitive to lighting and background conditions.

Deep learning, especially Convolutional Neural Networks [8], [9], has significantly improved recognition accuracy by learning features directly from images. Frameworks such as TensorFlow and Keras make training and deployment easier.

Google MediaPipe [2], [7] provides fast and robust hand detection and landmark tracking on ordinary hardware, which allows real-time performance without specialized devices. Many existing works, however, focus on a single language, run only as offline scripts, or lack a user-friendly web interface.

A. Sensor-Based Systems

Accurate but costly and intrusive, requiring gloves or wearable hardware.

B. Vision-Based Deep Learning Systems

Camera-only and affordable, but dependent on dataset quality and often lack a deployable interface.

C. Related Work on Indian Sign Language

Research on Indian Sign Language (ISL) is growing but still limited compared with American Sign Language. Many ISL studies are restricted to static alphabets and digits, use small datasets, and are evaluated only in controlled settings. Public datasets for ISL words and sentences are also fewer, which makes large-scale training difficult.

D. Research Gap

From the survey, the main gaps are: (i) dependence on special hardware, (ii) poor robustness to lighting and background, (iii) limited ISL support, and (iv) the absence of an end-to-end, web-deployable system. The proposed work addresses these gaps by combining MediaPipe-based detection, a CNN classifier, and a React frontend in a single pipeline.

Approach	Hardware	Cost	Real-time
Sensor / glove based	Gloves, sensors	High	Yes
Classical vision (SVM, k-NN)	Camera	Low	Limited
Proposed CNN + MediaPipe	Webcam	Low	Yes

Table I. Comparison of sign recognition approaches

VI. EXISTING SYSTEM

Current methods for communicating with sign language users depend mainly on human interpreters, written notes, or video relay services. Interpreters are not always available and add cost. Written communication is slow and may not suit users whose first language is sign language.

Automated tools that exist are often limited to a single sign language, need special hardware, or work only in controlled environments. Many of them do not provide a simple web interface for general users.

Limitations

- Dependence on human interpreters
- Costly or hardware-dependent solutions
- Limited support for Indian Sign Language
- No simple real-time web interface

VII. PROPOSED SYSTEM

The proposed system is a web-based application that recognizes hand signs in real time using a camera and displays the result as text. The camera captures video frames, OpenCV handles frame processing, and MediaPipe detects the hand region. The cropped hand region is passed to a trained CNN, which predicts the sign. The predicted label is sent to the React frontend through a REST API and shown to the user.

The overall flow of the system is:

Camera → Hand Gesture → OpenCV/MediaPipe → CNN Model → Sign Prediction → Text → React Frontend

Features

- Real-time hand detection and tracking
- Sign classification using a trained CNN
- Instant text display on the frontend
- REST API based communication
- Optional history storage using MongoDB

VIII. SYSTEM ARCHITECTURE

The architecture follows a client-server model with three layers.

The Presentation Layer is built with React.js (Vite), HTML, CSS, and JavaScript. It accesses the camera, shows the live video, and displays the translated text.

The Application Layer is a Python Flask service that receives frames from the frontend through REST endpoints, runs preprocessing, invokes the model, and returns the predicted sign.

The AI/Data Layer contains the trained CNN model, label mappings, and the optional MongoDB database for history. The model is loaded once at server start-up to ensure fast predictions.

The layered design keeps each component independent, so the model or the interface can be upgraded without changing the rest of the system.

IX. MODULE DESCRIPTION

1. Camera Input Module: captures live video frames from the webcam.
2. Hand Detection Module: uses MediaPipe to locate the hand and extract the region of interest.
3. Preprocessing Module: uses OpenCV to crop, resize, convert, and normalize frames.
4. Sign Recognition Module: a CNN classifies the processed hand image into a sign label.
5. API Module: the Flask REST API connects the model with the frontend.
6. Frontend Display Module: the React interface shows the live feed and translated text.
7. History Module (optional): stores recognized text and session data in MongoDB.

X. SYSTEM REQUIREMENTS

Component	Requirement
Processor	Intel Core i5 or equivalent
RAM	8 GB or higher
Camera	Standard webcam (720p)
Operating System	Windows / Linux / macOS
Software	Python 3.x, Node.js, VS Code
Libraries	OpenCV, MediaPipe, TensorFlow/Keras, Flask, React

Table II. Hardware and software requirements

The system is intentionally designed to run on ordinary laptops, so that it can be used without specialized equipment or GPU support during prediction.

XI. METHODOLOGY

A. Dataset Collection

Images of each sign are collected under varied lighting, backgrounds, and hand positions, from multiple signers where possible. Each image is labeled with its corresponding sign.

B. Preprocessing

Frames are cropped to the detected hand region, resized to a fixed size (for example 64×64), converted as required, and normalized. Data augmentation such as rotation, zoom, and brightness change is applied to improve generalization.

Hand detection uses MediaPipe, which tracks 21 landmarks per hand and provides a bounding region even when fingers partly overlap. The bounding region is expanded by a small margin so that the full hand is included, and the cropped region is used as the model input. This reduces the effect of the background and of the hand's position in the frame.

The dataset is divided into training (70%), validation (15%), and test (15%) sets. Augmentation techniques used are:

- Random rotation within a small angle range
- Zoom and width/height shifts
- Brightness and contrast variation
- Horizontal flip only for signs that remain valid when mirrored

C. Model Training

A CNN built with TensorFlow/Keras is trained on the preprocessed dataset. The dataset is split into training, validation, and test sets. The network uses convolution and pooling layers for feature extraction, followed by dense layers and a softmax output for classification. Dropout is applied to reduce overfitting.

Layer	Configuration
Input	64 × 64 image
Conv2D + ReLU	32 filters, 3 × 3
MaxPooling	2 × 2
Conv2D + ReLU	64 filters, 3 × 3
MaxPooling	2 × 2
Conv2D + ReLU	128 filters, 3 × 3
MaxPooling	2 × 2
Flatten	—
Dense + ReLU	128 units
Dropout	0.5
Dense + Softmax	N classes (signs)

Table III. CNN architecture used for sign classification

The model is compiled with the Adam optimizer [10] and categorical cross-entropy loss. Training uses early stopping on validation loss to prevent overfitting, and the best-performing weights are saved for deployment.

Parameter	Value
Optimizer	Adam
Learning rate	0.001
Batch size	32
Epochs	[XX] (early stopping)
Loss function	Categorical cross-entropy
Data split	70 / 15 / 15

Table IV. Training configuration

D. Real-Time Prediction

During use, each frame passes through hand detection and preprocessing, and the model returns the predicted sign with a confidence score. Predictions below a threshold are ignored to avoid false output.

E. Display

The predicted text is sent to the React frontend through the REST API and displayed instantly.

XII. ALGORITHM DESIGN

8. Start the camera and read a video frame.
9. Detect the hand using MediaPipe; if none is found, read the next frame.

10. Crop the hand region and preprocess it using OpenCV.
11. Pass the processed image to the trained CNN.
12. Obtain the predicted class and its confidence score.
13. If the confidence exceeds the threshold, map the class to its text label.
14. Send the label to the frontend through the REST API.
15. Display the text and repeat for the next frame.

XIII. API AND DATABASE DESIGN

The frontend and the model communicate through lightweight REST endpoints that exchange JSON data.

Endpoint	Purpose
POST /predict	Receives a frame; returns sign label and confidence
GET /history	Returns past recognized text (optional)
POST /history	Saves a recognized text entry (optional)
GET /health	Checks that the server and model are ready

Table V. REST API endpoints

When history is enabled, MongoDB stores a session collection containing the session ID, timestamp, recognized text, and confidence values. Indexing on timestamp allows fast retrieval, and no raw video is stored, which protects user privacy.

XIV. TECHNOLOGY STACK

Category	Technology
Frontend	React.js (Vite), HTML, CSS, JavaScript
Computer Vision	OpenCV
Hand Detection	MediaPipe
Deep Learning	TensorFlow / Keras (CNN)
Backend	Python, Flask
Database	MongoDB (optional)
API	REST
Dev Tool	VS Code

Table VI. Technologies used in the system

XV. USER INTERFACE DESIGN

The interface is simple and accessible. A live camera panel shows the user's hand with detection feedback, and a text panel displays the recognized sign in large, readable text. Controls allow the user to start or stop the camera and clear the output. The layout is responsive and works on desktops and laptops, with a consistent theme across the application.

XVI. IMPLEMENTATION

The AI pipeline is implemented in Python. OpenCV captures and processes frames, MediaPipe detects the hand, and the Keras model performs classification. The Flask server exposes REST endpoints that accept frames from the frontend and return predictions as JSON.

The React frontend, created with Vite, accesses the webcam, sends frames to the API at regular intervals, and renders the returned text. The application is tested locally and can be deployed on a cloud platform for wider access.

Use Case Walkthrough

A user opens the web application and grants camera permission. The live video appears on the screen. When the user shows a sign, MediaPipe detects the hand and the cropped region is sent to the server. The CNN predicts the sign, and the recognized text appears below the video within a fraction of a second. The user can then clear the output or continue with the next sign. This simple flow makes the system easy to use for people with no technical background.

Performance Considerations

Prediction speed depends on frame size, model size, and network delay. Keeping the CNN small, loading it once at start-up, and sending reduced-size frames at a fixed interval keep the response smooth. Frame smoothing, where a sign is accepted only after it is predicted consistently across several frames, further reduces flicker and false outputs.

XVII. TESTING

Functional testing checks that camera access, hand detection, prediction, and text display work correctly. Performance testing measures prediction time per frame. Robustness testing evaluates behavior under different lighting, backgrounds, and signers.

A. Unit Testing

Preprocessing, model prediction, and API endpoints are tested individually.

B. Integration Testing

The complete flow from camera to frontend display is tested end to end.

C. Model Evaluation

The model is evaluated on unseen test data using accuracy, precision, recall, F1-score, and a confusion matrix. Accuracy is the ratio of correct predictions to total predictions. Precision is TP / (TP + FP), recall is TP / (TP + FN), and the F1-score is the harmonic mean of precision and recall, where TP, FP, and FN denote true positives, false positives, and false negatives.

Test case	**Expected result**
Camera access allowed	Live video is displayed
Hand in frame	Hand is detected
No hand in frame	No prediction is shown
Clear known sign	Correct text is displayed
Low confidence sign	Prediction is ignored
Server not running	Error message is shown

Table VII. Sample functional test cases

XVIII. RESULTS

The system was implemented and tested in a local environment. It detected hands from the live camera feed and displayed the corresponding text on the frontend with low delay.

The CNN model achieved a test accuracy of [XX]% on the evaluation dataset, with an average prediction time of [XX] ms per frame. [Insert your measured results here.]

Metric	Value
Test accuracy	[XX]%
Precision	[XX]%
Recall	[XX]%
F1-score	[XX]%
Avg. prediction time	[XX] ms/frame

Table VIII. Model performance metrics

The confusion matrix shows that most errors occur between signs with similar hand shapes. Adding more varied training images and applying stronger augmentation reduced such confusion. Performance was best under good lighting with a plain background and dropped slightly in dim or cluttered scenes.

The interface was found to be simple and easy to use, and the REST-based design allowed smooth communication between the frontend and the model. Overall, the results show that the system meets its objective of converting hand signs into text in real time.

XIX. DISCUSSION

The results show that a lightweight CNN combined with MediaPipe-based hand localization can achieve real-time recognition on ordinary hardware. Using MediaPipe to isolate the hand greatly reduces background noise, which is one of the main causes of error in older vision-based systems.

Several factors influence accuracy. Lighting quality, camera resolution, and the distance of the hand from the camera affect detection. Signs with very similar shapes are more likely to be confused. Variation between signers, such as hand size and speed, also affects predictions. Training with diverse data and applying a confidence threshold helped reduce wrong outputs.

Compared with sensor-based systems, the proposed approach is far cheaper and easier to deploy, although sensor-based methods may remain more precise for complex gestures. Compared with traditional vision methods, the CNN learns features automatically and generalizes better across users.

Challenges and Solutions

Challenge	Solution
Varying lighting	Brightness augmentation, normalization
Cluttered background	MediaPipe hand cropping
Similar-looking signs	More samples, deeper features
Prediction flicker	Confidence threshold, frame smoothing
Latency	Load model once, send reduced-size frames

Table IX. Challenges faced and solutions applied

XX. DEPLOYMENT

The Flask API and the trained model can be hosted on a cloud platform, while the React build is served as a static site. Because inference runs on the server, users only need a browser and a camera. For lower latency, the frontend sends reduced-size frames at a fixed interval rather than every frame, and the server returns only the predicted label and confidence.

XXI. APPLICATIONS

- Education: helping teachers and students communicate in classrooms
- Healthcare: assisting doctors and patients at hospitals and clinics
- Public services: banks, railway stations, and help desks
- Workplaces: inclusive communication with hearing-impaired employees

- Learning tool: helping beginners practise sign language

XXII. PRIVACY AND ETHICAL CONSIDERATIONS

Since the system uses a live camera, privacy is an important concern. Video frames are processed only for prediction and are not stored by default. Users are asked for camera permission before use, and any stored history contains only recognized text. The system should be trained on data collected with consent and should represent signers of different ages, skin tones, and signing styles to reduce bias.

XXIII. ADVANTAGES

- Works with an ordinary webcam; no gloves or sensors needed
- Real-time recognition and display
- Low cost and easy to use
- Web-based, with no installation for end users
- Modular design that is easy to extend

XXIV. LIMITATIONS

- Accuracy depends on lighting, background, and camera quality
- Recognizes only the signs included in the training dataset
- Supports isolated signs rather than full sentences
- Differences in signing style may reduce accuracy
- Requires a device with a camera and a stable connection

XXV. FUTURE SCOPE

Future work includes recognition of continuous signs and full sentences using sequence models such as LSTM or Transformers, an expanded Indian Sign Language vocabulary, text-to-speech output, and text-to-sign conversion for two-way communication. A mobile application, multi-language text output, and larger, more diverse datasets can further improve usability and accuracy.

XXVI. CONCLUSION

This project presents an AI-based system that recognizes hand signs through a camera and converts them into text in real time. By combining OpenCV, MediaPipe, a CNN model, and a React frontend, the system offers an affordable and accessible way to bridge the communication gap between sign language users and others. Although limited to a defined set of signs, it provides a strong foundation for future improvements toward complete, sentence-level sign language translation.

ACKNOWLEDGMENT

The authors thank Dr. Kamalaveni V for her guidance and support throughout this work.

REFERENCES

[1] OpenCV Documentation, https://docs.opencv.org

[2] Google MediaPipe Hands Documentation, https://developers.google.com/mediapipe

[3] TensorFlow and Keras Documentation, https://www.tensorflow.org

[4] React Documentation, https://react.dev

[5] Flask Documentation, https://flask.palletsprojects.com

[6] MongoDB Documentation, https://www.mongodb.com/docs

[7] F. Zhang et al., “MediaPipe Hands: On-device Real-time Hand Tracking,” arXiv:2006.10214, 2020.

[8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proc. IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[9] A. Krizhevsky, I. Sutskever, and G. Hinton, “ImageNet classification with deep convolutional neural networks,” NIPS, 2012.

[10] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” ICLR, 2015.

[11] [Add the sign language recognition papers used in your literature survey]