Technical Report: Real-Time Indian Sign Language (ISL) Alphabet Recognizer

Project Version: 1.0
Date: August 8, 2025
Author: Deepak K

1. Executive Summary

This report details the design, implementation, and performance of a real-time computer vision system for recognizing 23 static signs of the Indian Sign Language (ISL) alphabet. The primary objective was to create an accessible and accurate tool to facilitate communication for the deaf and hard-of-hearing community.

The system employs a four-stage machine learning pipeline: **(1)** automated data acquisition of 23,000 images using OpenCV; **(2)** high-fidelity feature engineering using Google's MediaPipe framework to extract normalized hand landmark coordinates; **(3)** model training using a RandomForestClassifier from scikit-learn; and **(4)** deployment of a real-time inference engine via a user-friendly Streamlit web application.

The resulting model achieved an outstanding **overall accuracy of 99.63%** on the unseen test set. A detailed analysis of per-class metrics (precision, recall, F1-score) and a 10-fold cross-validation confirms the model's robustness and high-performance across all signs. The final application demonstrates stable, real-time performance, successfully translating ISL gestures captured from a standard webcam into corresponding English letters.

2. System Architecture and Methodology

The project is logically divided into a sequential pipeline, where the output of each stage serves as the input for the next.

Stage 1: Data Acquisition

- **Script:** collect_images.py
- **Process:** A custom dataset was created to ensure sufficient data for training. The script utilizes OpenCV to access the system webcam.
- **Dataset Parameters:**
  - **Classes:** 23 static ISL alphabet signs ('J' and 'Z', which involve motion, were excluded).
  - **Volume: 1,000 images** were captured for each of the 23 classes, resulting in a raw dataset of **23,000 images**.
- **Data Diversity Strategy:** To build a model robust to user handedness, the script prompts the user to switch hands after capturing the first 500 images for each sign. This simple yet effective technique ensures the model learns from both left- and right-handed representations.
- **Output:** Images are stored in a structured directory format: isl_data/<SIGN_LABEL>/<image_number>.jpg.

Stage 2: Feature Engineering

- **Script:** create_dataset_isl.py
- **Core Technology:** Google's **MediaPipe Hands** library is the cornerstone of the feature extraction process. Instead of using raw pixels, the system converts each image into a compact and meaningful numerical representation.
- **Feature Vector Construction:**
  1. MediaPipe identifies **21 key landmarks** (joints, fingertips) for each detected hand.
  2. The system is designed to handle **up to two hands**, crucial for signs that require both.
  3. A fixed-length feature vector of **84 dimensions** is created for every image (PER_HAND_LEN of 42 for the left hand, followed by 42 for the right).
  4. **Normalization:** To ensure the model is invariant to hand size and position in the frame, the (x, y) coordinates of each landmark are normalized relative to the wrist (landmark 0) of that specific hand.
  5. **Handling Missing Hands:** If only one hand is detected (or none), the corresponding 42-dimension slot in the vector is filled with zeros. This creates a consistent input shape for the classifier.
- **Output:**
  - data_isl.pickle: A file containing two objects: a list of the 84-dimension feature vectors (data) and a parallel list of their corresponding string labels (labels).

    - processing_errors.log: A log file documenting images where MediaPipe failed to detect any hands.

Stage 3: Model Training & Evaluation

- **Script:** train_classifier_isl.py
- **Model Selection:** A **RandomForestClassifier** was chosen. This ensemble model is highly effective for this type of tabular, high-dimensional data as it is robust to overfitting and can capture complex non-linear relationships between features. The model was configured with n_estimators=200, creating a forest of 200 decision trees.
- **Data Preprocessing:**
    - **Scaling:** Before training, the feature data (X) is standardized using StandardScaler. This process removes the mean and scales each feature to unit variance, which is critical for the optimal performance of many machine learning algorithms.
    - **Data Splitting:** The dataset is divided into an **80% training set** and a **20% testing set**. The split is stratified (stratify=y), ensuring that the proportion of each sign is the same in both the train and test sets, which is crucial for a balanced evaluation.
- **Model Persistence:** The trained RandomForestClassifier object and the fitted StandardScaler object are bundled together and saved to a single file, model_isl.p, using joblib. This ensures that the exact same scaling transformation can be applied during real-time inference.

Stage 4: Real-Time Inference Engine

- **Script:** recogniser_streamlit.py
- **Framework:** A web application is built using **Streamlit**, providing a clean, interactive, and user-friendly interface that runs directly in the browser.
- **Real-Time Pipeline:**
    1. The application loads the saved model and scaler from model_isl.p.
    2. It accesses the webcam feed using OpenCV.
    3. For each frame, it performs the **exact same feature extraction process** as in Stage 2 (using MediaPipe to generate an 84-dimension vector).
    4. The extracted vector is scaled using the loaded scaler and fed into the model's .predict() method.
- **Prediction Smoothing:** To prevent the predicted letter from flickering rapidly due to minor hand movements between frames, a **deque** (a fixed-size queue) is used to store the last few predictions. The final letter displayed to the user is determined by a **majority vote** within this buffer, resulting in a significantly more stable and readable output.

3. Performance Metrics and Results Analysis

The model's performance was rigorously evaluated on the 20% test set, which it had never seen during training.

3.1 Overall Accuracy

The primary metric for success is accuracy, which measures the proportion of correct predictions.

- **Test Accuracy: 99.63%**

This exceptional result indicates that the model correctly identifies the ISL sign in 996 out of every 1000 unseen instances.

3.2 Per-Class Performance: Classification Report

The classification report provides a granular breakdown of performance for each individual sign, evaluating precision, recall, and F1-score.

Class	**Precision**	**Recall**	**F1-Score**	**Support (Samples)**
A	1.00	1.00	1.00	200
B	1.00	1.00	1.00	200
C	0.99	0.99	0.99	194
D	1.00	1.00	1.00	200
E	1.00	1.00	1.00	199
F	1.00	1.00	1.00	200
G	1.00	1.00	1.00	200
H	1.00	1.00	1.00	199
I	1.00	1.00	1.00	200
K	1.00	1.00	1.00	200
L	1.00	1.00	1.00	200
M	1.00	0.99	0.99	199

N	0.99	1.00	1.00	200
O	1.00	1.00	1.00	199
P	1.00	1.00	1.00	200
Q	1.00	1.00	1.00	200
S	1.00	1.00	1.00	200
T	1.00	1.00	1.00	200
U	1.00	1.00	1.00	200
W	1.00	1.00	1.00	200
X	0.99	1.00	1.00	199
Y	1.00	1.00	1.00	200
Z	1.00	0.99	0.99	198
Avg/Total	**1.00**	**1.00**	**1.00**	**4586**

- **Interpretation:** The model demonstrates near-perfect precision and recall for almost every class. The minor dips (e.g., 0.99 for 'C', 'M', 'N', 'X', 'Z') are statistically insignificant and still represent elite performance. This indicates the features extracted by MediaPipe are highly discriminative.

3.3 Confusion Matrix Analysis

The confusion matrix visualizes the model's predictions, showing where, if any, errors were made.

- **Analysis:** The matrix exhibits an overwhelmingly strong diagonal line. Each cell on the diagonal represents the number of correct predictions for that class. The off-diagonal cells are almost entirely dark/zero, indicating a very low number of misclassifications. For example, the cell at the intersection of the 'M' row and 'N' column would show how many times an 'M' was incorrectly predicted as an 'N'. The near-absence of color in these areas confirms the model's high fidelity.

3.4 Cross-Validation

To ensure the model's performance is not just a result of a "lucky" train-test split, a 10-fold cross-validation was performed.

- **Process:** The entire dataset was split into 10 parts. The model was trained 10 times, each time using 9 parts for training and 1 part for validation.
- **Result:** The average accuracy across the 10 folds was **99.55% ± 0.09%**.
- **Interpretation:** This result is extremely strong. The low standard deviation (±0.09%) indicates that the model's performance is highly consistent and stable, regardless of how the training and validation data is partitioned. This gives very high confidence in the model's ability to generalize to new, unseen data.

3.5 Data Quality Analysis

An analysis of processing_errors.log reveals that **1074 out of 23,000 images (4.67%)** were discarded because MediaPipe could not detect a hand.

- **Common Failure Cases:** A review of the log shows a higher concentration of errors for signs like 'C', 'S', 'Z', and 'H'.
- **Hypothesis:** This is likely due to hand self-occlusion. In signs like 'C' and 'S' (a fist), fingers obscure the palm and other landmarks, making it difficult for the vision model to confidently identify a hand. This is a known limitation of landmark detection models and highlights an area for future data collection improvement (e.g., capturing these signs from slightly different angles).

4. Conclusion and Recommendations

This project successfully developed a high-accuracy, real-time ISL alphabet recognizer. The combination of a large, custom dataset, robust MediaPipe feature extraction, and a well-tuned Random Forest classifier has proven to be an extremely effective solution. The final Streamlit application provides a practical and accessible proof-of-concept.

Recommendations for Future Work:

1. **Expand to Dynamic Gestures:** The most significant next step is to move beyond static signs. This would involve:
   - **Data:** Collecting video sequences of words and phrases.
   - **Modeling:** Employing sequence models like **LSTMs (Long Short-Term Memory networks), GRUs, or Transformers**, which are designed to understand temporal patterns. The input would be a sequence of the 84-dimension vectors generated in this project.
2. **Enhance Dataset Robustness:** To improve real-world performance, the dataset should be expanded by collecting images under more challenging conditions:
   - **Varied Lighting:** Bright, dim, and uneven lighting.
   - **Cluttered Backgrounds:** Non-uniform backgrounds to test the model's focus on the hands.
   - **Diverse Signers:** Data from a wider range of individuals of different ages and skin tones.
3. **Mobile Deployment:** For maximum accessibility, the model should be deployed to a mobile application.
   - **Optimization:** The model could be converted to a more lightweight format like **TensorFlow Lite** or **ONNX**.
   - **Application:** A native iOS/Android app could use the phone's camera for a truly portable communication aid.
4. **Explore Alternative Models:** While Random Forest performed exceptionally well, exploring other architectures could yield further improvements. **Gradient Boosting models (like XGBoost or LightGBM)** are strong candidates and often outperform Random Forest on tabular data.