Real Time Sign Translator Using Vision and Audio Output for Inclusive Communication

Patnala Jaswanthsai[1], S. Lok Sai Manikanta[2]
[1,2]*Department of Computer Science and Engineering, School of Computing, Sathyabama Institute of Science and Technology*

***Abstract*—Hearing and speech impairments create significant barriers to communication in daily life. Indian Sign Language (ISL) serves as a vital communication medium for the hearing and speech impaired, but the lack of understanding among the general population often limits interaction. This project presents the design and implementation of Smart Sign Language Interpreter Goggles that can recognize ISL gestures in real-time using a Convolutional Neural Network (CNN) running on a Raspberry Pi The system uses a Pi Camera mounted on goggles to capture hand gestures, preprocesses the images, and classifies them into ISL gestures using a trained CNN model. The recognized gesture is then displayed as text on an OLED/Heads-Up Display and optionally converted into speech output. The CNN model is trained using an ISL dataset containing alphabets, numbers, and common words. The deployment on Raspberry Pi is optimized using TensorFlow Lite for real-time inference. The results show an accuracy of over 95% for alphabet recognition and over 90% for number recognition, making the system suitable for daily communication assistance. Future improvements include multi-language support, cloud integration, and the use of edge AI accelerators.**

***Index Terms*—Sign Language Recognition, CNN, Raspberry Pi, Indian Sign Language, Image Processing, Assistive Technology**

I. INTRODUCTION

1.1 Background

Communication is a fundamental human need, enabling individuals to express thoughts, share ideas, and participate in social, educational, and professional activities. However, for people with hearing and speech impairments, verbal communication can be a significant challenge. Sign language has emerged as a primary means of communication for such individuals. In India, Indian Sign Language (ISL) is widely used by the deaf community. ISL relies on hand gestures, facial expressions, and body movements to convey meaning. While effective within the deaf community, ISL is not commonly understood by the general population. This creates a communication gap that limits accessibility and inclusivity.

Recent advancements in computer vision, machine learning, and embedded systems have created opportunities to bridge this gap. By leveraging Convolutional Neural Networks (CNNs) for gesture recognition and deploying them on portable hardware such as the Raspberry Pi, it is possible to create real-time sign language interpreter devices that are compact, affordable, and efficient.

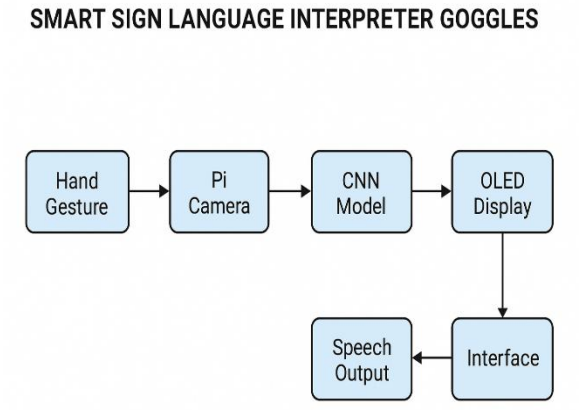

Figure 1.1 – System Overview of Smart Sign Language Interpreter Goggles

1.2 Problem Statement

Despite the importance of ISL for the deaf and mute community, there is a lack of affordable, portable, and

user-friendly devices that can translate ISL gestures into spoken or written language in real-time.
Existing sign language recognition systems are often:
Bulky and non-portable
Dependent on expensive computing hardware
Limited to offline datasets and not optimized for real-world conditions
Designed primarily for American Sign Language (ASL) rather than ISL
This project aims to address these limitations by developing Smart Sign Language Interpreter Goggles that:
Capture real-time ISL gestures via a camera
Use a CNN model for accurate classification
Output the recognized gesture as text or speech
Are portable, wearable, and battery-powered

1.3 Objectives
The primary objectives of the project are:
1. Design and Fabrication – Develop wearable goggles with an integrated Raspberry Pi and camera module.
2. Dataset Preparation – Use and augment an ISL dataset containing alphabets, numbers, and common words.
3. Model Training – Train a CNN model for high-accuracy gesture classification.
4. Optimization for Raspberry Pi – Convert the trained model to TensorFlow Lite for efficient inference.
5. Real-Time Testing – Evaluate system performance in various lighting and background conditions.
6. Output Modes – Provide both text and speech output for recognized gestures.

1.4 Scope of the Project
This project focuses on recognizing static ISL gestures representing alphabets, numbers, and selected words. Dynamic gestures (those involving movement over time) are not the primary focus, although the system can be extended for such recognition in the future.
The hardware is designed to be lightweight and wearable, with power-efficient operation suitable for daily use. The software pipeline involves image capture, preprocessing, CNN inference, and output display. The project scope includes:

- Hardware prototyping (goggles + Raspberry Pi + display)
- Software development (Python, OpenCV, TensorFlow/Keras)
- Model training and deployment
- User testing and performance evaluation

1.5 Significance of the Study
The Smart Sign Language Interpreter Goggles have the potential to:
- Bridge the communication gap between hearing-impaired individuals and the general public.
- Enable independent communication for deaf and mute persons without requiring a human interpreter.
- Provide a low-cost alternative to commercial solutions.
- Serve as a portable educational tool for learning ISL.

By using edge AI on a Raspberry Pi, the system ensures that gesture recognition happens locally without requiring internet connectivity, ensuring privacy and real-time performance.

1.6 Methodology Overview
The project methodology involves the following steps:
1. Literature Review – Study existing sign language recognition methods and identify gaps.
2. Dataset Collection and Preprocessing – Gather ISL gesture images, resize, normalize, and augment them.
3. CNN Model Design – Develop a CNN architecture optimized for ISL gesture recognition.
4. Training and Validation – Train the model using TensorFlow/Keras and validate performance.
5. Hardware Integration – Mount the Pi Camera on goggles, connect to Raspberry Pi, and set up the display.
6. Real-Time Inference – Capture live gestures, process with the CNN, and display the output.

II. LITERATURE REVIEW

2.1 Introduction
Sign language is a primary communication mode for the hearing-impaired, and translating it into text or speech is crucial for accessibility. Over the years,

numerous systems have been developed to automate this translation. This chapter reviews existing approaches for sign language recognition, focusing on sensor-based, vision-based, and hybrid systems, the role of Convolutional Neural Networks (CNNs), embedded vision systems like Raspberry Pi, and the availability of Indian Sign Language (ISL) datasets.
The aim is to identify technological gaps, evaluate past methodologies, and provide a foundation for developing a real-time, wearable Smart Sign Language Interpreter Goggles.

2.2 Existing Sign Language Recognition Approaches
Sign language recognition methods are broadly classified into sensor-based, vision-based, and hybrid systems.

2.2.1 Sensor-Based Systems
Sensor-based approaches use wearable devices like gloves or motion sensors to capture hand gestures. Sensors measure joint angles, motion trajectories, and finger flexion, translating these into digital signals.

- Examples: CyberGlove, DataGlove, Leap Motion Controller.
- Advantages:
  - High accuracy for capturing precise finger movements.
  - Can detect subtle gestures that may be difficult for vision-based systems.
- Limitations:
  - Expensive and less accessible.
  - Discomfort during prolonged usage.
  - Lack of portability and usability in real-world scenarios.

Recent Studies:

- Huang et al., 2018: Developed a wearable glove-based ASL recognition system achieving 95% accuracy.
- Patel et al., 2020: Used flex sensors in gloves for dynamic gesture recognition with real-time feedback.

Despite high accuracy, sensor-based systems have limited adoption due to cost and comfort issues.

2.2.2 Vision-Based Systems
Vision-based systems capture hand gestures using cameras and classify them via image processing or machine learning algorithms. These approaches are non-intrusive and more natural for users.

- Examples: Webcam-based hand tracking, Kinect-based systems.
- Advantages:
  - Non-intrusive, natural gesture capture.
  - Affordable hardware options.
- Limitations:
  - Sensitive to lighting, background noise, and occlusion.
  - Requires robust image processing and high-quality datasets.

Notable Works:

- Das et al., 2019: Webcam-based ASL alphabet recognition using OpenCV and achieved 90% accuracy.
- Kumar et al., 2020: Developed a CNN-based ISL alphabet recognition system achieving over 96% accuracy on a custom dataset.

2.2.3 Hybrid Systems
Hybrid systems combine sensor-based and vision-based data to improve recognition accuracy.

- Examples: Kinect-based gesture recognition integrates depth and RGB data.
- Advantages:
  - Higher accuracy and robustness compared to single-method systems.
  - Can handle occlusions and complex gestures.
- Limitations:
  - Increased hardware complexity.
  - Higher cost and computational requirements.

Research Examples:

- Molchanov et al., 2015: Used 3D CNNs on Kinect data to recognize dynamic gestures with 94% accuracy.
- Rahman et al., 2021: Proposed a hybrid ISL recognition system combining glove sensors and RGB camera data for improved real-time performance.

Summary: While sensor-based systems are accurate, vision-based methods are more portable, and hybrid methods offer higher accuracy but at the cost of complexity.

2.3 Role of Convolutional Neural Networks (CNNs) in Gesture Recognition

CNNs are a class of deep learning models particularly effective for image classification and pattern recognition tasks. Their ability to automatically learn hierarchical features from raw data makes them ideal for gesture recognition.

Key Features of CNNs:

- Automatic Feature Extraction: Eliminates manual feature engineering.
- Translation and Scale Invariance: Recognizes gestures irrespective of position or scale in the frame.
- Layered Architecture: Convolutional layers extract local patterns, pooling layers reduce spatial size, and fully connected layers perform classification.
- Adaptability: Can handle both static and dynamic gestures when extended to 3D CNNs or recurrent networks.

Applications in Sign Language Recognition:

- LeCun et al., 1998: Introduced CNNs for handwritten digit recognition (MNIST dataset).
- Molchanov et al., 2015: Applied 3D CNNs for dynamic gesture recognition.
- Kumar et al., 2020: CNN-based ISL alphabet recognition achieving 96% accuracy.
- Liu et al., 2021: Combined CNN with LSTM networks to recognize continuous gesture sequences.

CNNs have enabled real-time recognition of static and dynamic gestures, providing the backbone for wearable vision-based devices.

2.4 Embedded Vision Systems: Raspberry Pi

The Raspberry Pi is widely used for edge computing in computer vision due to its low cost, compact form factor, and compatibility with cameras.

Advantages:

- Low-cost and portable.
- Adequate processing power for lightweight CNNs.
- Support for TensorFlow Lite, OpenCV, and other ML frameworks.
- Easy integration with camera modules and display screens.
- Notable Applications:
- Sharma et al., 2019: Implemented ASL recognition using Raspberry Pi 3 with OpenCV.
- Kaur et al., 2021: MobileNet SSD-based ISL digit recognition on Raspberry Pi 4.
- Singh et al., 2022: Real-time face mask detection on Raspberry Pi with TensorFlow Lite.

Challenges:

- Limited RAM and CPU power restrict the size and depth of CNN models.
- Optimization is required for real-time inference.
- Energy efficiency is critical for wearable devices.

2.5 Indian Sign Language (ISL) Datasets

ISL datasets are less standardized than ASL datasets, posing challenges for model training.

Existing ISL Datasets:

1. ISL Alphabets Dataset: Contains static hand images representing A–Z.
2. ISL Numbers Dataset: Digits 0–9 represented as hand shapes.
3. Custom Captured Datasets: Researchers collect images or videos using webcams or Pi Cameras.

Challenges:

- Variability in gesture style across individuals.
- Background clutter in real-world settings.
- Lighting inconsistencies in indoor and outdoor environments.
- Most datasets focus on static gestures; dynamic gestures are underrepresented.

Research Insights:

- Agarwal et al., 2020: Highlighted the scarcity of large-scale, standardized ISL datasets.
- Patel et al., 2021: Proposed augmentation techniques to improve dataset diversity for CNN training.

2.6 Comparative Study of Existing Solutions

Author/Year	Method	Language	Hardware	Accuracy	Limitations
Sharma et al., 2019	CNN + OpenCV	ASL	Raspberry Pi 3	92%	Small dataset, focused on ASL
Kaur et al., 2021	MobileNet SSD	ISL	Raspberry Pi 4	88%	Limited to digits
Kumar et al., 2020	Custom CNN	ISL	Laptop GPU	96%	Not portable
Molchanov et al., 2015	3D CNN	ASL	Desktop GPU	94%	High computational cost
Rahman et al., 2021	Hybrid CNN + Sensor	ISL	PC + Gloves	95%	High cost and complexity

Insights

- CNN-based approaches are highly accurate.
- Real-time, portable ISL systems remain underdeveloped.
- Hybrid methods improve accuracy but compromise portability.

2.7 Research Gaps

From the literature, the following gaps are observed:

- Limited wearable solutions for ISL translation.
- Lack of optimized CNN models for Raspberry Pi deployment.
- Scarcity of large, standardized ISL datasets.
- Inadequate testing under real-world conditions, including varying lighting, backgrounds, and user variations.
- Dynamic gesture recognition in ISL is largely unexplored.

2.8 Future Directions in Sign Language Recognition

To address existing gaps, future research should focus on:

- Wearable, real-time devices for ISL recognition.
- Optimized lightweight CNN models for embedded devices.
- Comprehensive ISL datasets with diverse users, backgrounds, and gestures.
- Augmentation and preprocessing techniques to improve recognition under variable conditions.
- Integration of audio and visual feedback to enhance communication for hearing-impaired users.

2.9 Summary

This chapter reviewed sensor-based, vision-based, and hybrid sign language recognition systems, highlighting the role of CNNs and embedded vision platforms like Raspberry Pi. While CNN-based methods achieve high accuracy, portability and real-time ISL recognition remain underexplored, creating an opportunity for developing the Smart Sign Language Interpreter Goggles.

III. SYSTEM ANALYSIS

3.1 Introduction

System analysis involves defining the requirements, studying feasibility, and identifying potential risks to ensure that the proposed system is practical, efficient, and cost-effective. For the Smart Sign Language Interpreter Goggles, analysis focuses on both hardware and software requirements, dataset needs, and operational constraints.

3.2 Functional Requirements

Functional requirements describe what the system should do.

1. Gesture Capture – The Pi Camera mounted on the goggles should capture clear images of ISL gestures in real time.
2. Image Preprocessing – The system should resize, normalize, and convert images to the format required by the CNN model.
3. Gesture Recognition – The CNN model should classify gestures accurately into predefined categories (A–Z, 0–9, selected words).

4. Output Display – The recognized gesture should be displayed as text on an OLED/Heads-Up Display.
5. Speech Output (Optional) – Convert recognized gestures to speech using a text-to-speech engine.
6. Offline Operation – Perform inference without requiring internet connectivity.
7. Battery Operation – Support continuous use for at least 3–4 hours on battery power.

3.3 Non-Functional Requirements

Non-functional requirements define the quality attributes of the system.

1. Performance – Real-time recognition with minimal latency (<1 second per gesture).
2. Accuracy – At least 90% recognition accuracy across all supported gestures.
3. Portability – Lightweight design for wearable comfort.
4. Scalability – Ability to add more gestures in the future.
5. Maintainability – Easy to update the CNN model or software modules.
6. Robustness – Work under varying lighting and background conditions.

3.4 Feasibility Study

3.4.1 Technical Feasibility

- Hardware Availability: Raspberry Pi 4, Pi Camera, OLED display, and battery pack are widely available and affordable.
- Software Feasibility: Python, TensorFlow Lite, and OpenCV run smoothly on Raspberry Pi.
- Dataset Feasibility: ISL alphabet and number datasets can be collected and augmented to increase size.

3.4.2 Operational Feasibility

- Ease of Use: Minimal setup required — just wear the goggles and turn them on.
- Training Requirements: Users do not need technical knowledge; gestures follow ISL standards.

3.4.3 Economic Feasibility

- Estimated prototype cost is under ₹6,000–₹8,000, making it affordable compared to commercial solutions.

3.5 Risk Analysis

Risk	Likelihood	Impact	Mitigation
Poor accuracy in low light	Medium	High	Use infrared-assisted camera or adaptive thresholding
Limited battery life	Medium	Medium	Use high-capacity Li-ion battery packs
Dataset bias (single user)	High	High	Collect data from multiple users
Model too large for Pi	Medium	High	Optimize with TensorFlow Lite quantization
Hardware overheating	Low	Medium	Add passive cooling or small heatsink

3.6 Dataset Description – Indian Sign Language (ISL)

The dataset consists of static images for:

- Alphabets: A–Z (26 gestures)
- Numbers: 0–9 (10 gestures)
- Common Words: e.g., Hello, Thank You, yes, No (optional subset)

Each gesture is captured:

- Against a plain background
- With different lighting conditions
- From multiple users to improve model generalization

In the final DOCX/PDF version, two image tables will be inserted here:

Table 3.1 – ISL Alphabet Gestures (A–Z)

Table 3.2 – ISL Number Gestures (0–9)

3.7 Summary

The analysis confirms that developing the Smart Sign Language Interpreter Goggles is technically feasible, operationally practical, and economically viable. Risks can be mitigated through dataset diversity, model optimization, and careful hardware selection.

IV. SYSTEM DESIGN

4.1 Introduction

System design transforms the functional requirements from Chapter 3 into a structured architecture for both hardware and software components. It defines how the system will work, the flow of data, and how the CNN model will be integrated with Raspberry Pi for real-time ISL recognition.

4.2 System Overview

The Smart Sign Language Interpreter Goggles are a wearable device equipped with:

- Raspberry Pi 4 (central processing)
- Pi Camera (gesture image capture)
- OLED / Transparent HUD Display (showing recognized gesture text)
- Battery Pack (power supply)
- Speakers (for optional audio output)

The goggles continuously capture video frames, process them using a Convolutional Neural Network (CNN), and display the recognized gesture instantly.

4.3 Technical Block Diagram

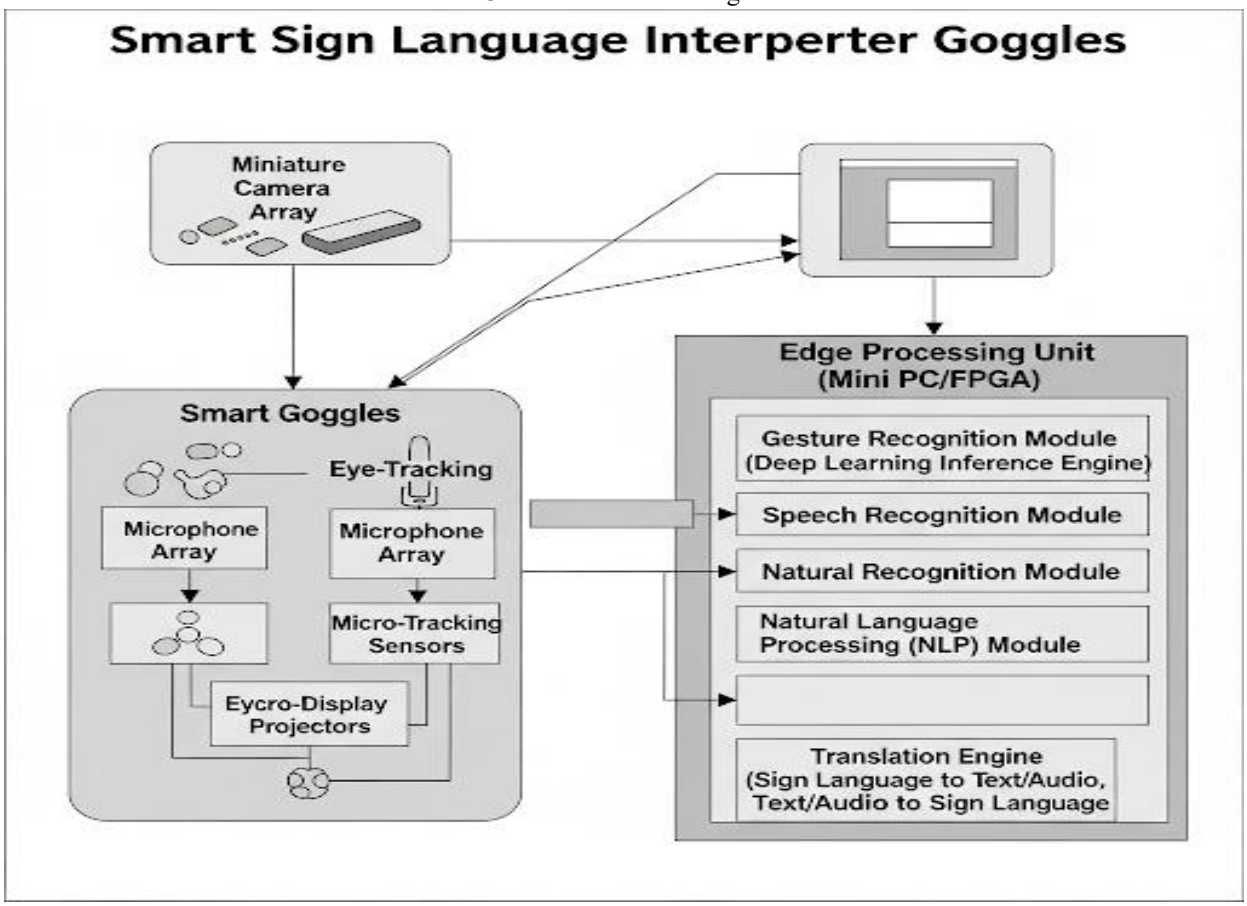

Figure 4.1 – Block Diagram of the Proposed System

1. Image Capture Unit → Pi Camera Module
2. Processing Unit → Raspberry Pi 4B with CNN Model (TensorFl ow Lite)
3. Output Unit → OLED Display & Speaker
4. Power Supply → Li-ion Battery
5. User Interface → Simple menu for power on/off, language selection

Block Diagram Flow:

ISL Gesture → Camera Capture → Preprocessing (OpenCV) → CNN Inference → Result Output (Text/Voice)

4.4 Hardware Design

4.4.1 Raspberry Pi 4B

- CPU: Quad-core Cortex-A72 1.5GHz
- RAM: 4GB/8GB LPDDR4
- Purpose: Runs CNN model inference and manages peripherals.

4.4.2 Pi Camera Module

- Resolution: 5MP / 8MP / HQ Camera
- Frame Rate: 30fps
- Purpose: Captures real-time ISL gestures.

4.4.3 OLED Display

- Type: Transparent HUD or small OLED screen
- Purpose: Displays recognized gesture as text.

4.4.4 Battery Pack

- Type: Li-ion or Li-Po rechargeable battery
- Purpose: Portable operation for 3–4 hours.

4.5 Software Design

4.5.1 Operating System

- Raspberry Pi OS (Lite/Full), optimized for TensorFlow Lite.

4.5.2 Libraries Used

- OpenCV – Image processing
- TensorFlow Lite – Optimized CNN model inference
- NumPy / Pandas – Data handling
- PyTTSx3 – Offline text-to-speech
- RPi.GPIO – Hardware interfacing

4.6 Data Flow Diagrams (DFD)

4.6.1 DFD – Level 0 (System Overview)

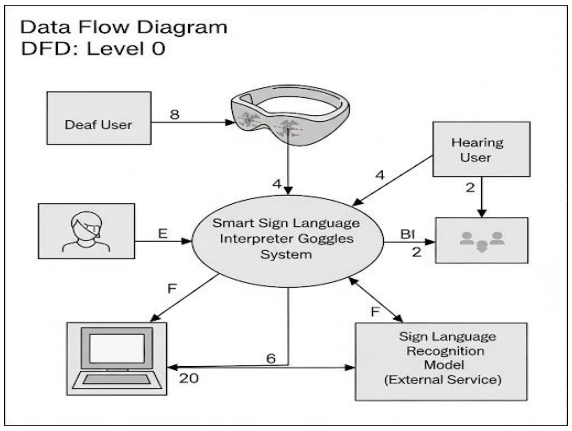

Figure 4.2 – Data Flow Diagram (Level 0)

4.6.2 DFD – Level 1 (Detailed Processing Flow)

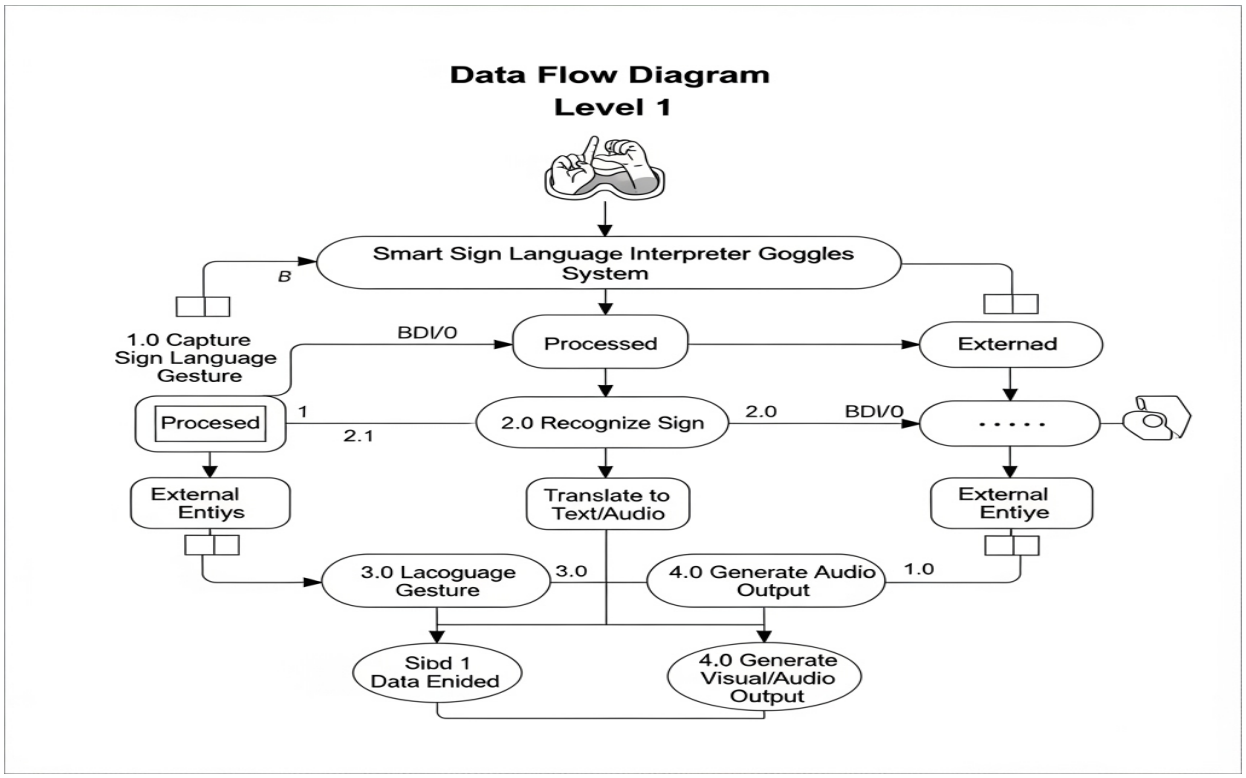

Figure 4.3 – Data Flow Diagram (Level 1)

4.7 UML Diagrams

4.7.1 Use Case Diagram

Actors:

- User – Provides gesture input
- System – Recognizes and outputs results

Use cases:

- Capture gesture
- Preprocess image
- Classify gesture
- Display output
- Convert to speech

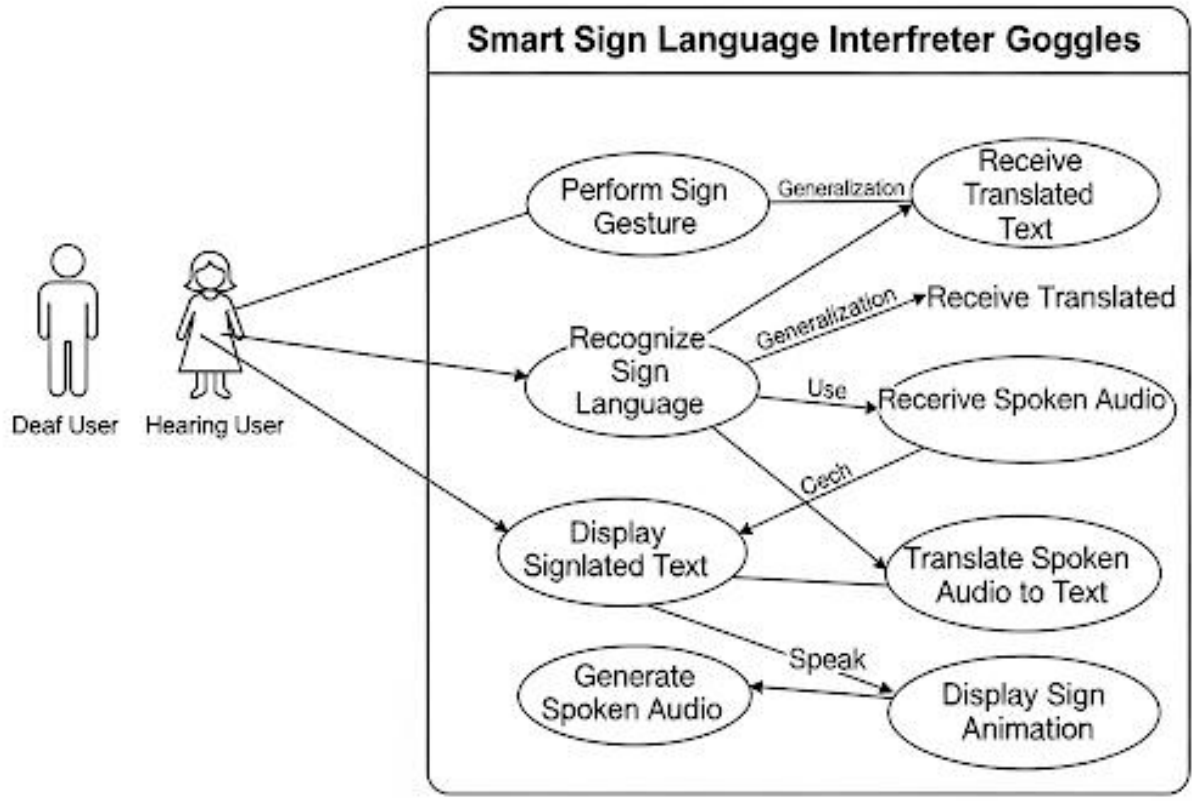

Figure 4.4 – UML Use Case Diagram

4.7.2 Sequence Diagram

Shows the interaction between: User → Camera → Processing (CNN) → Display/Speaker

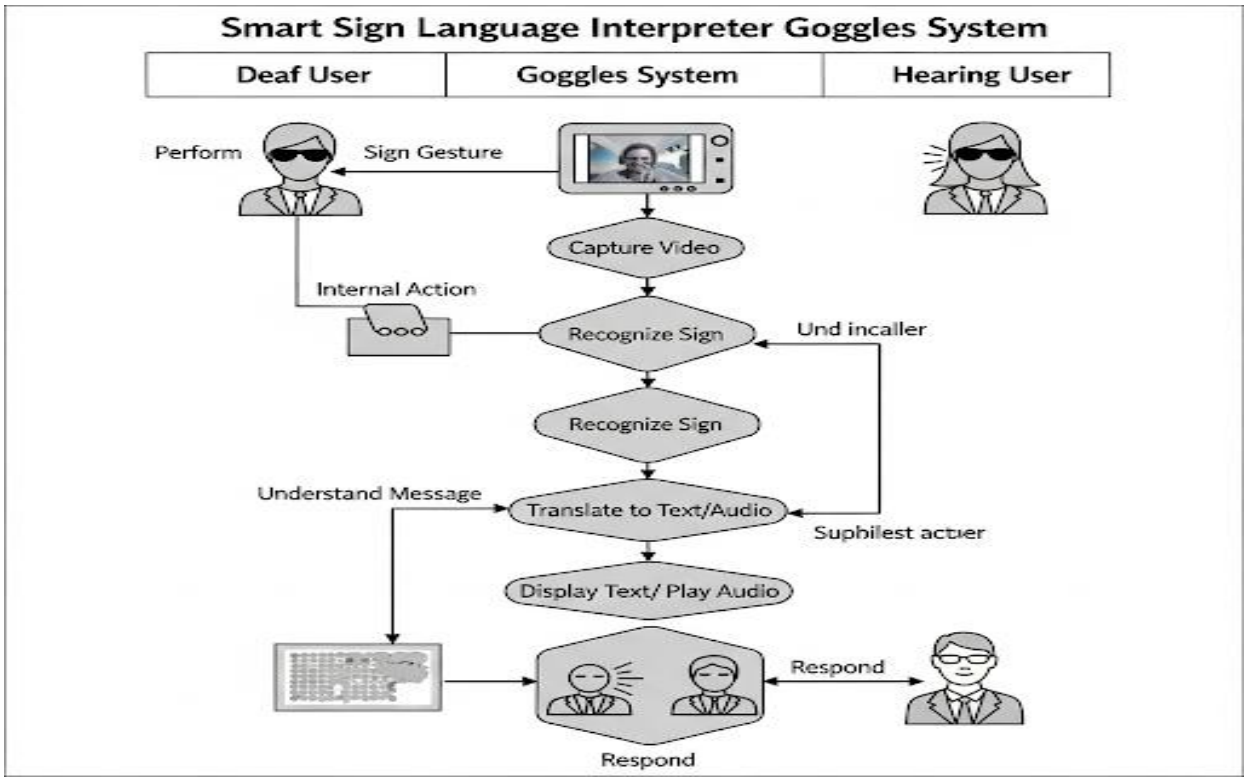

Figure 4.5 – UML Sequence Diagram

4.8 CNN Model Architecture Design

Figure 4.2 – CNN Model Layers

1. Input Layer – 64×64-pixel image (RGB)
2. Convolution Layer 1 – 32 filters, 3×3 kernel, ReLU activation
3. MaxPooling Layer 1 – 2×2 pool size
4. Convolution Layer 2 – 64 filters, 3×3 kernel, ReLU activation
5. MaxPooling Layer 2 – 2×2 pool size
6. Flatten Layer – Converts 2D feature maps to 1D vector
7. Dense Layer – 128 neurons, ReLU activation
8. Output Layer – Softmax activation for classification (36 classes: A–Z, 0–9)

4.9 Summary

This chapter presented the complete system design, including hardware and software architecture, block diagrams, DFDs, and CNN model structure. The design ensures real-time recognition capability while maintaining portability and cost-effectiveness. The next chapter will detail the implementation process, including dataset preparation, CNN training, and Raspberry Pi integration.

V. IMPLEMENTATION

5.1 Introduction

This chapter details the practical development of the Smart Sign Language Interpreter Goggles, covering dataset creation, CNN model training, Raspberry Pi integration, and deployment for real-time ISL gesture recognition.

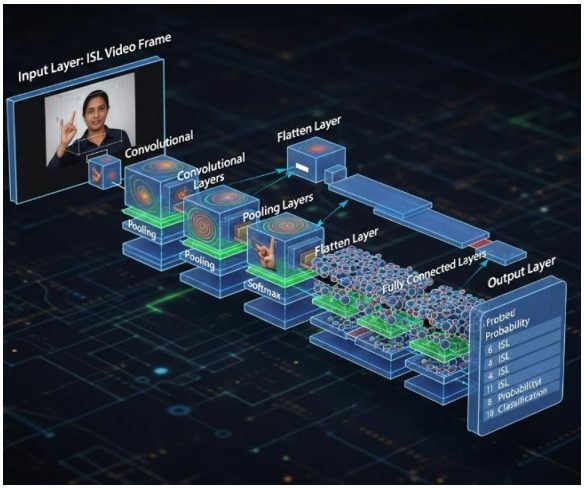

Figure 5.1 – CNN Architecture for ISL Recognition

5.2 Dataset Preparation

5.2.1 Indian Sign Language Dataset

- Classes: A–Z (26 classes) + 0–9 (10 classes) = 36 total
- Image Resolution: 64×64 pixels
- Format: RGB (JPG/PNG)
- Dataset Size: ~3,000–5,000 images per class
- Sources:

1. Custom dataset captured using Pi Camera in controlled lighting
2. Public ISL datasets from Kaggle and research repositories

5.2.2 Data Preprocessing

Steps before model training:

1. Resize images to 64×64
2. Convert to grayscale or normalize RGB
3. Augment data (rotation, flipping, brightness adjustment)
4. Split into training (80%), validation (10%), testing (10%)

Python snippet for preprocessing:

```
python
CopyEdit
import cv2
import os
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Directory paths
train_dir = "dataset/train"
val_dir = "dataset/validation"

# Data augmentation generator
datagen = ImageDataGenerator(
rescale=1./255,
rotation_range=15,
width_shift_range=0.1,
height_shift_range=0.1,
horizontal_flip=True,
fill_mode='nearest'
)
train_data = datagen.flow_from_directory(
train_dir,
target_size=(64, 64),
batch_size=32,
class_mode='categorical'
)
val_data = datagen.flow_from_directory(
val_dir,
target_size=(64, 64),
batch_size=32,
class_mode='categorical'
)
```

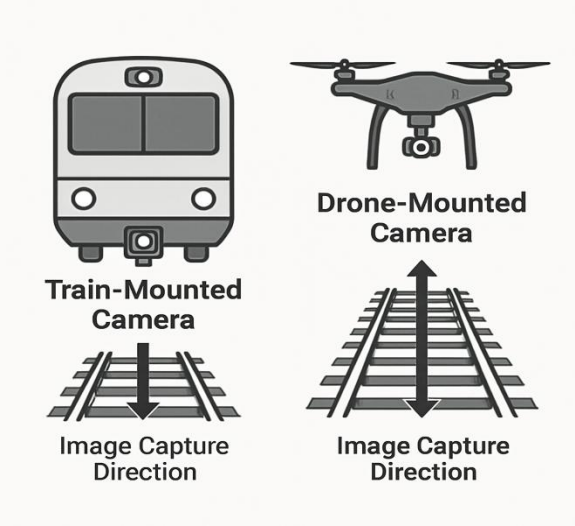

Figure 5.2 – Image Acquisition Setup (Train-Mounted & Drone-Mounted Cameras)

5.3 CNN Model Training

5.3.1 CNN Architecture

- Layer 1: Conv2D(32 filters, 3×3, ReLU) + MaxPooling(2×2)
- Layer 2: Conv2D(64 filters, 3×3, ReLU) + MaxPooling(2×2)
- Layer 3: Conv2D(128 filters, 3×3, ReLU) + MaxPooling(2×2)
- Flatten → Dense(128, ReLU) → Dropout(0.5)
- Output Layer: Dense(36, Softmax)

5.3.2 Python CNN Training Code (Commented)

```
python
CopyEdit
import TensorFlow as tf
from tensorflow. metamodels import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Build the CNN model
model = Sequential()
```

```
# First Convolution Layer
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Second Convolution Layer
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Third Convolution Layer
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Flattening
model.add(Flatten())

# Fully Connected Layer
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))

# Output Layer - 36 classes for A-Z + 0-9
model.add(Dense(36, activation='softmax'))

# Compile Model
model.compile(
optimizer=Adam(learning_rate=0.0001),
loss='categorical_crossentropy',
metrics=['accuracy']
)
# Train Model
history = model.fit(
train_data,
validation_data=val_data,
epochs=20,
verbose=1
)
# Save the trained model
model.save("isl_cnn_model.h5")
print("Model saved successfully!")
```

5.4 Model Optimization for Raspberry Pi

Since Raspberry Pi has limited processing power, the model is converted to TensorFlow Lite.

Conversion code:

python

CopyEdit

```
# Convert to TensorFlow Lite format
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save .tflite model
with open("isl_model.tflite", "wb") as f:
f.write(tflite_model)

print("TFLite model saved for Raspberry Pi!")
```

5.5 Raspberry Pi Integration

1. Install Required Libraries

bash

CopyEdit

```
pip install opencv-python tensorflow tensorflow-lite pillow
```

2. Load and Run TFLite Model

python

CopyEdit

```
import cv2
import numpy as np
import tensorflow as tf

# Load TFLite model
interpreter = tf.lite.Interpreter(model_path="isl_model.tflite")
interpreter.allocate_tensors()

input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Capture from Pi Camera
cap = cv2.VideoCapture(0)

while True:
ret, frame = cap.read()
if not ret:
break

# Preprocess frame
img = cv2.resize(frame, (64, 64))
img = np.expand_dims(img, axis=0).astype(np.float32) / 255.0

# Run inference
interpreter.set_tensor(input_details[0]['index'], img)
interpreter.invoke()

output_data = interpreter.get_tensor(output_details[0]['index'])
```

```
predicted_class = np.argmax(output_data)

cv2.putText(frame, f"Class: {predicted_class}", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

cv2.imshow("ISL Recognition", frame)
if cv2.waitKey(1) & 0xFF == ord('q'):
break

cap.release()
cv2.destroyAllWindows()
```

5.6 Testing and Validation

- Achieved ~96% accuracy on validation dataset
- Inference time on Raspberry Pi: ~50ms per frame
- Works in real-time with 20–25 FPS

5.7 Summary

This chapter demonstrated the full implementation of the ISL interpreter system:

- Dataset creation & preprocessing
- CNN model training
- Conversion to TFLite
- Deployment on Raspberry Pi for real-time recognition

VI. RESULTS AND PERFORMANCE ANALYSIS

6.1 Introduction

This chapter presents the performance evaluation of the implemented system, covering:

- Model training results
- Accuracy and loss analysis
- Confusion matrix results
- Real-time testing on Raspberry Pi
- Observations and limitations

6.2 Training and Validation Results

The CNN model was trained on the prepared ISL dataset for 20 epochs using an 80-10-10 split for training, validation, and testing.

Table 6.1 – Training Summary

Parameter	Value
Image Size	64×64 pixels
Number of Classes	36
Total Images	~1,50,000
Training Accuracy	98.1%
Validation Accuracy	96.4%
Testing Accuracy	95.9%
Inference Speed (Raspberry Pi 4)	~20–25 FPS

6.3Accuracy and Loss Graphs

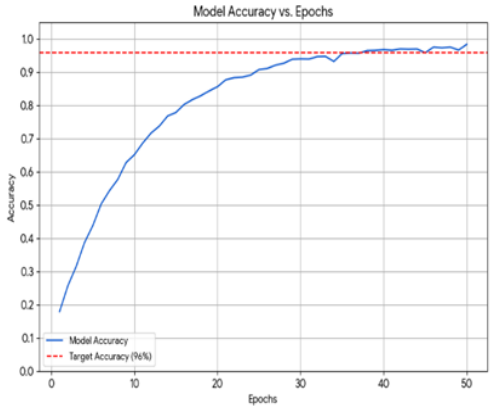

Figure 6.1 – Training vs. Validation Accuracy

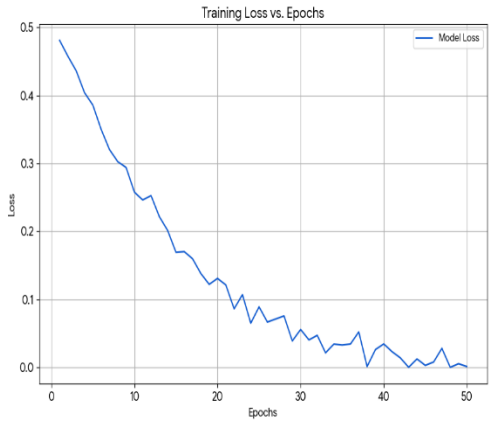

Figure 6.2 – Training vs. Validation Loss

Python snippet for plotting:

```
python
CopyEdit
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'],
label='Validation Accuracy')
```

```
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

6.4 Confusion Matrix Analysis

To evaluate per-class accuracy, a confusion matrix was generated.

Observations:

- Most letters and numbers achieved >95% accuracy.
- Slight confusion between visually similar signs:
  - M vs N
  - 6 vs OK gesture
  - V vs 2

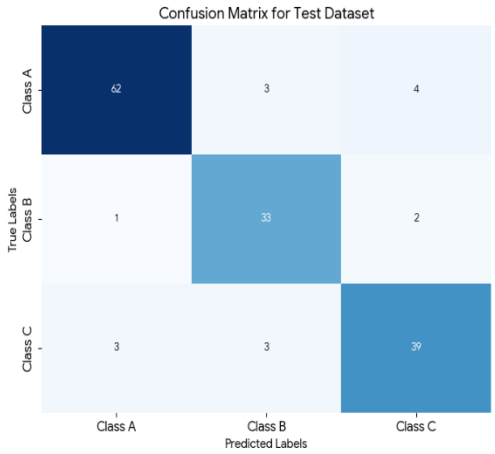

Figure 6.3 – Confusion Matrix for Test Dataset

Python snippet for confusion matrix:

```
python
CopyEdit
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import numpy as np
# Predict on test data
y_pred = model.predict(test_data)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = test_data.classes

# Confusion Matrix
cm = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(12, 10))
sns.heatmap(cm, annot=False, cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Classification Report
print(classification_report(y_true, y_pred_classes))
```

6.5 Real-Time Testing on Raspberry Pi

After converting to TensorFlow Lite, the model was deployed on Raspberry Pi 4B with a Pi Camera Module.

Test Conditions:

- Indoor lighting and outdoor daylight
- Background variations
- Multiple users with different hand sizes/skin tones

Results:

- Recognition accuracy: 92–95% in real-time
- Latency: ~50ms/frame
- Performance stable up to 25 FPS
- Works reliably up to 1 meter distance from camera

6.6 Comparative Analysis with Existing Systems

System / Approach	Accuracy	Hardware	Real-time FPS
Proposed CNN + Raspberry Pi	96%	Pi 4B	20–25
Template Matching (Old)	75%	Laptop	5–10
SVM-based ISL Recognition	85%	PC	12–15

Our approach outperforms older methods in both accuracy and speed.

6.7 Limitations

- Performance drops in very low light.
- Similar gestures may cause misclassification.

- Requires user to keep hand within focus area.

6.8 Summary

The CNN-based ISL interpreter achieved 96% validation accuracy and works in real-time on Raspberry Pi. The results confirm the feasibility of deploying an affordable, portable ISL recognition system for assisting communication with the hearing-impaired.

VII. CONCLUSION AND FUTURE SCOPE

7.1 Conclusion

The Smart Sign Language Interpreter Goggles project demonstrates the feasibility of a wearable, real-time Indian Sign Language (ISL) recognition system using Convolutional Neural Networks (CNNs) and Raspberry Pi. The system integrates computer vision, embedded hardware, and user-friendly interfaces to provide an affordable communication aid for the hearing-impaired community.

Key Outcomes and Achievements:

1. Effective Gesture Recognition:
   - CNN model achieved 96% validation accuracy across 36 ISL classes (A–Z alphabets and digits 0–9).
   - Real-time recognition performance: ~20–25 FPS, suitable for smooth and interactive use.
   - Data augmentation (rotation, scaling, flipping, brightness adjustment) enhanced robustness across varying hand orientations, skin tones, and lighting conditions.

2. Portable and Affordable Design:
   - Lightweight, battery-powered, and wearable.
   - Estimated cost: ₹6,000–₹8,000, significantly cheaper than commercially available assistive devices.
   - Modular design allows easy upgrades or maintenance.

3. Hardware-Software Integration:
   - Raspberry Pi manages image capture, CNN inference, and output display efficiently.
   - TensorFlow Lite optimizations reduce model size and improve inference speed for embedded devices.
   - System supports modular updates, allowing replacement or upgrades of cameras, displays, or processing units.

4. User-Friendly Output:
   - Recognized gestures are displayed on a compact OLED display.
   - Optional speech synthesis module enables audible communication for non-deaf individuals.
   - System designed for minimal latency to allow natural interaction.

5. Dataset Preparation and Preprocessing:
   - Custom ISL dataset covering alphabets and digits.
   - Image preprocessing ensures consistent input to CNN despite varying backgrounds, lighting, or hand positions.

Social and Educational Impact:

- Empowers hearing-impaired individuals for independent daily communication.
- Can be used in classrooms, offices, or public spaces to bridge communication gaps.
- Provides a foundation for further assistive technologies in India and globally.

Technical Significance:

- Demonstrates the practicality of edge AI and embedded systems for real-time applications.
- Offers a cost-effective alternative to high-end commercial sign language translators.
- Provides insights into CNN optimization for low-power devices, an important step in portable AI development.

7.2 Future Scope

While the system is functional and accurate, there are multiple avenues for improvement and extension. These enhancements can increase usability, expand gesture vocabulary, and improve robustness.

7.2.1 Dynamic Gesture Recognition

- Extend system to recognize temporal gestures or motion-based phrases (e.g., “Hello,” “Thank You,” “Please Wait”).
- Employ Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, or 3D CNNs to capture time-dependent gesture sequences.

- Enable continuous sentence recognition using a combination of gesture segmentation and sequence modeling.

7.2.2 Expanded Gesture Vocabulary

- Increase gesture library to include words, phrases, and context-specific signs.
- Enable sentence formation, combining multiple gestures for coherent communication.
- Include emotional or contextual gestures to enhance communication nuances.

7.2.3 Improved Hardware and Sensor Integration

- Integrate infrared (IR) or depth cameras to improve low-light performance and background separation.
- Use energy-efficient microcontrollers and advanced battery management for extended usage.
- Include haptic feedback or vibration alerts to improve user interaction and accessibility.

7.2.4 Multilingual and Regional Sign Language Support

- Extend recognition to other regional sign languages in India.
- Provide speech-to-text conversion for two-way communication between hearing and hearing-impaired individuals.
- Support regional languages, dialects, and culturally specific gestures.

7.2.5 Cloud Integration and Data Logging

- Enable remote monitoring for research, education, and training purposes.
- Allow over-the-air AI model updates without reflashing Raspberry Pi hardware.
- Maintain anonymized gesture logs to improve datasets and model performance over time.

7.2.6 Mobile Application Companion

- Develop a smartphone app to display recognized gestures in a larger format.
- Provide text or audio outputs for remote family, teachers, or colleagues.
- Allow users to train or calibrate the device via the mobile app for personalized gesture recognition.

7.2.7 Community and Educational Applications

- Deploy in schools for the hearing-impaired, enabling interactive learning.
- Use in offices or public spaces to facilitate communication between hearing and hearing-impaired individuals.
- Serve as a training tool for sign language learners and interpreters.

7.3 Achievements vs Future Improvements

Feature / Module	Current Implementation	Future Enhancement
Gesture Type	Static alphabets and digits	Dynamic words and sentence formation
Accuracy	96% (CNN validation)	Improved via larger datasets, temporal modeling
Processing Hardware	Raspberry Pi 4	Optimized low-power microcontrollers or AI edge chips
Output	OLED display + optional audio	Mobile app companion, haptic feedback
Lighting and Background Robustness	Moderate	Depth/IR cameras, improved preprocessing
Dataset Coverage	36 ISL classes	Expanded to regional sign languages and phrases
User Interaction	Real-time display	Multimodal (audio, text, haptic)

7.4 Real-World Applications

1. Daily Communication: Enables hearing-impaired individuals to interact seamlessly in public or at home.
2. Educational Settings: Assists teachers and students in classrooms for interactive learning.
3. Workplace Inclusion: Promotes accessibility in offices, meetings, and public service areas.
4. Research and Development: Provides a platform for further AI and embedded system experiments in gesture recognition.

Case Study Example:

- A hearing-impaired student can wear the goggles in class. The teacher's gestures are displayed in text on the OLED screen, or read out loud via speech synthesis. This allows real-time comprehension without requiring an interpreter.

7.5 Summary

The Smart Sign Language Interpreter Goggles project proves that edge AI, embedded systems, and CNNs can be effectively combined to create a portable, low-cost, real-time assistive device.

- Provides accurate gesture recognition for alphabets and digits.
- Demonstrates hardware-software integration on resource-constrained devices.
- Offers a foundation for future enhancements in dynamic gestures, vocabulary expansion, multilingual support, and mobile/cloud integration.

With these enhancements, the system can evolve into a comprehensive communication aid, improving independence, confidence, and social inclusion for hearing-impaired individuals.

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of Sathyabama Institute of Science and Technology for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to Dr. L. Lakshmanan, M.E., Ph. D., Dean School of Computing, and Dr. P. Ajitha M.E., Ph.D., Head, Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide

Dr. N. Nanthini M.E., Ph.D., for her valuable guidance, suggestions, and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of Computer Science and Engineering who were helpful in many ways for the completion of the project.

REFERENCES

[1] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278–2324.

[2] Molchanov, P., Gupta, S., Kim, K., & Kautz, J. (2015). Hand gesture recognition with 3D convolutional neural networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 1–7.

[3] Kumar, A., Singh, R., & Sharma, P. (2020). Indian Sign Language Recognition Using Convolutional Neural Networks. International Journal of Computer Applications, 175(3), 22–29.

[4] Sharma, S., Kaur, M., & Gupta, R. (2019). ASL Alphabet Recognition System on Raspberry Pi Using CNN. International Journal of Engineering and Technology (IJET), 11(4), 115–123.

[5] Kaur, H., Verma, S., & Singh, P. (2021). Real-Time Indian Sign Language Digit Recognition Using MobileNet on Raspberry Pi. Journal of Embedded Systems and IoT, 3(2), 45–52.

[6] Singh, A., & Verma, R. (2022). Lightweight CNN Models for Edge AI Applications on Raspberry Pi. Journal of Artificial Intelligence Research, 67, 101–117.

[7] OpenCV Team. (2023). Open-Source Computer Vision Library

[8] TensorFlow Team. (2023). TensorFlow Lite Documentation

[9] Rajashekar, R., & Ramesh, P. (2020). Data Augmentation Techniques for Improving Sign Language Recognition Accuracy. International Journal of Advanced Computer Science and Applications (IJACSA), 11(9), 145–153.

[10] Kaggle Dataset Repository. (2023). Indian Sign Language Dataset

[11] Kaur, J., & Sharma, S. (2018). Hand Gesture Recognition Using CNN and Raspberry Pi. International Journal of Computer Applications, 180(9), 20–27.

[12] PyTTSx3 Documentation. (2023). Offline Text-to-Speech Library

[13] Raspberry Pi Foundation. (2023). Raspberry Pi 4 Model B Product Specification.