



Department of Electronics and Communication Engineering,

Amrita School of Engineering,

Amrita Vishwa Vidyapeetham, Coimbatore-641112

***19CSE348-MACHINE LEARNING TERM PROJECT***

***Sign Language Detector:  
A Machine Learning Approach***

**WORK SUBMITTED BY:**

**Chittesh ST  
Joshua John Britto  
Umaa Maheswaran V  
Varun Kaleeswaran B**

## 1. ABSTRACT

This project showcases a system that can recognize hand gestures in real time using MediaPipe and a simple neural network built with PyTorch. Gesture images for five actions ("Hello," "Please," "Thank you," "OK," and "Thumbs up") were captured using a webcam. The system detected hand landmarks and turned them into simplified feature sets, which were used to train a neural network. The training process resulted in a model that could recognize these gestures with high accuracy.

Once trained, the model was tested in a live video setup, where it successfully detected hand gestures and displayed the corresponding labels on the screen. This project combines computer vision and machine learning to create an interactive system, with potential applications in areas like sign language interpretation or hands-free device control.

## 2. INTRODUCTION

Sign language is a vital form of communication for millions of individuals who are deaf or hard of hearing, providing a powerful way to express thoughts, emotions, and information through structured hand gestures, facial expressions, and body language. Despite its importance, sign language interpretation poses challenges due to the intricate nature

of gestures and the need for real-time processing. Automated sign language recognition systems can bridge the communication gap between sign language users and the wider community, enhancing accessibility and inclusivity.

Recent advances in computer vision and deep learning have opened up new possibilities for real-time sign language detection. By leveraging technologies such as convolutional neural networks (CNNs), Mediapipe hand tracking, and PyTorch, we can develop robust solutions capable of recognizing sign language gestures from video input. These systems rely on identifying key hand landmarks and translating those into meaningful gestures using machine learning models trained on annotated datasets.

This project focuses on the development of a real-time sign language recognition system that utilizes computer vision techniques to detect and classify hand gestures. The system uses the Mediapipe framework for hand landmark detection, which captures 21 key points of the hand to extract x and y coordinates. These coordinates are then processed to train a neural network capable of recognizing specific sign language gestures. The model is trained on a custom dataset containing five distinct classes of gestures, aiming to provide an effective solution for recognizing commonly used signs such as

"Hello", "Please", "Thank You", "OK", and "Thumbs Up." By implementing this solution, we aim to demonstrate the potential of using deep learning models for sign language detection, thereby contributing to the development of accessible technologies that can improve communication for individuals with hearing impairments.

### 3. LITERATURE REVIEW

Sign language recognition (SLR) has been an active research area, with various approaches focusing on bridging communication gaps for the deaf-mute community through technology. Numerous techniques have been explored to improve the accuracy and efficiency of recognizing sign gestures using computer vision and machine learning.

1. **Ashok K. Sahoo, Gouri S. Mishra, and Kiran K. Ravulakollu (2014):**

In their comprehensive review on sign language recognition, the authors outline various methodologies for recognizing sign gestures using both static and dynamic models. Their study emphasizes the challenges in data acquisition, feature extraction, and classification, focusing particularly on Indian Sign Language (ISL) due to the scarcity of research in this domain

2. **I.A. Adeyanju, O.O. Bello, and M.A. Adegboye (2021):**

This critical review highlights the advancements in using intelligent systems for sign language recognition. The authors present a bibliometric analysis of publications in the field, indicating a significant increase in research efforts, especially in vision-based SLR systems. The review focuses on using deep learning models, such as Convolutional Neural Networks (CNNs), to enhance recognition accuracy and reduce execution time.

3. **Refat Khan Pathan et al. (2023):**

This study explores the use of multi-headed convolutional neural networks (CNNs) to improve the recognition of American Sign Language (ASL). By combining traditional image processing techniques with hand landmark detection, the authors achieved a high recognition accuracy of 98.98%. The research emphasizes the need for robust models that can handle noisy real-world data, thus reducing dependency on controlled environments for training

These studies collectively highlight the evolution of SLR systems from traditional computer vision techniques to the integration of advanced deep learning

frameworks. The use of CNNs, data augmentation, and efficient preprocessing methods are emphasized as critical factors in improving system accuracy and robustness. The literature suggests that while significant progress has been made, challenges remain in achieving real-time performance in diverse, uncontrolled environments.

#### 4. MODEL DESCRIPTION

The proposed system for Sign Language Recognition (SLR) leverages state-of-the-art computer vision and deep learning techniques to efficiently classify hand gestures in real-time. This model aims to bridge communication gaps by recognizing predefined sign language gestures using video input captured through a webcam. The model is built upon a combination of MediaPipe for hand landmark detection and a deep learning framework using PyTorch for gesture classification.

##### 1. Data Collection and Preprocessing

The foundation of any machine learning model lies in the quality of its training data. In this project, video frames are captured using a standard webcam to create a custom dataset of sign language gestures. The data collection involves capturing images for five different gesture classes, each containing 1,000 samples.

##### Data Collection Pipeline

- Data Directory Structure: Each

gesture class is saved in a separate folder, with image frames stored as .jpg files.

- Hand Gesture Capture: A webcam is used to capture hand gestures in real-time. The captured frames are saved for each class after the user initiates data collection by pressing a key. This ensures that the data is user-labeled and organized into different categories.

##### Data Preprocessing

**Hand Landmark Detection:** The captured images are processed using the MediaPipe framework, which detects 21 key landmarks of the hand (including finger joints and wrist). This step extracts crucial features that represent the gesture.

**Normalization of Coordinates:** The landmark coordinates are normalized by subtracting the minimum x and y values. This ensures that the model remains invariant to hand position within the frame.

**Feature Extraction:** The normalized x and y coordinates of the 21 landmarks (resulting in 42 features) are extracted and stored as input features for the model.

**Data Augmentation:** Techniques like random scaling, rotation, and translation can be applied to increase the variability of the dataset, though not explicitly used in this implementation.

```

DATA_DIR = './data'
if not os.path.exists(DATA_DIR):
    os.makedirs(DATA_DIR)

number_of_classes = 5
dataset_size = 1000

cap = cv2.VideoCapture(0)
for j in range(number_of_classes):
    if not os.path.exists(os.path.join(DATA_DIR, str(j))):
        os.makedirs(os.path.join(DATA_DIR, str(j)))

    print('Collecting data for class {}'.format(j))

    done = False
    while True:
        ret, frame = cap.read()
        cv2.putText(frame, 'Ready? Press "Q" ! :)', (100, 50), cv2.FONT_HERSHEY_SIMPLEX, 1.3, (0, 255, 0), 3,
                    cv2.LINE_AA)
        cv2.imshow('frame', frame)
        if cv2.waitKey(25) == ord('q'):
            break

    S
    counter = 0
    while counter < dataset_size:
        ret, frame = cap.read()
        cv2.imshow('frame', frame)
        cv2.waitKey(25)
        cv2.imwrite(os.path.join(DATA_DIR, str(j), '{}.jpg'.format(counter)), frame)

        counter += 1

cap.release()
cv2.destroyAllWindows()

```

## 2. Model Architecture

The deep learning model used for sign language classification is a simple yet effective feedforward neural network (FNN) built using the PyTorch framework. The architecture is designed to take in the 42 extracted hand landmark features and output a classification label for the detected gesture.

### Model Components

**Input Layer:** The input layer receives a vector of length 42, representing the normalized x and y coordinates of the hand landmarks.

**Hidden Layer:** The first hidden layer consists of 64 neurons, followed by a ReLU activation function, which introduces non-

linearity to the model.

The ReLU activation function is chosen for its ability to avoid the vanishing gradient problem and to accelerate convergence.

**Output Layer:** The output layer consists of n neurons (where n is the number of gesture classes, in this case, 5). A softmax function is applied to convert the outputs into probabilities for each class.

```

# Define model parameters
input_size = fixed_length
hidden_size = 64
output_size = len(np.unique(labels_encoded))

# Define your PyTorch model
class SimpleNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

```

### 3. Model Training and Optimization

The model is trained using a supervised learning approach, with the dataset split into training (80%) and testing (20%) sets.

The training process involves:

- **Loss Function:** Cross-Entropy Loss is used, as it is suitable for multi-class classification problems.
- **Optimizer:** The Adam optimizer is employed due to its efficiency in handling sparse gradients and adapting the learning rate.

- **Training Loop:**

- The training loop iterates through the dataset for a specified number of epochs (1000 epochs in this case).
- For each batch, the model performs forward propagation to make predictions, calculates the loss, performs backpropagation, and updates the weights.

```
##Model train
# Load data
data_dict = pickle.load(open('./data.pickle', 'rb'))
data = data_dict['data']
labels = data_dict['labels']

# Determine the fixed length for each sample (e.g., 42 coordinates if 21 Landmarks with x and y each)
fixed_length = 42 # Adjust this based on your Landmarks

# Pad or truncate data to fixed length
data_padded = []
for sample in data:
    if len(sample) < fixed_length:
        sample += [0] * (fixed_length - len(sample))
    else:
        sample = sample[:fixed_length]
    data_padded.append(sample)

# Convert to numpy array
data_padded = np.array(data_padded)
labels = np.asarray(labels)

# Encode labels to numeric values
label_encoder = LabelEncoder()
labels_encoded = label_encoder.fit_transform(labels)

# Split data
x_train, x_test, y_train, y_test = train_test_split(data_padded, labels_encoded, test_size=0.2, shuffle=True, stratify=labels_encoded)
```

 1

```

# Split data
x_train, x_test, y_train, y_test = train_test_split(data_padded, labels_encoded, test_size=0.2, shuffle=True, stratify=labels_encoded)

# Convert data to PyTorch tensors
x_train_tensor = torch.tensor(x_train, dtype=torch.float32)
x_test_tensor = torch.tensor(x_test, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.long)
y_test_tensor = torch.tensor(y_test, dtype=torch.long)

# Define model parameters
input_size = fixed_length
hidden_size = 64
output_size = len(np.unique(labels_encoded))

# Define your PyTorch model
class SimpleNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

```

```

# Load model if exists, otherwise train
model_path = 'model.pth'
model = SimpleNN(input_size, hidden_size, output_size)

if os.path.exists(model_path):
    print("Loading model from file...")
    model.load_state_dict(torch.load(model_path))
else:
    # Define loss function and optimizer
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters())

    # Training loop
    num_epochs = 1000
    for epoch in range(num_epochs):
        outputs = model(x_train_tensor)
        loss = criterion(outputs, y_train_tensor)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    # Save the model's state dictionary
    torch.save(model.state_dict(), model_path)
    print("Model trained and saved to file.")

# Evaluation
model.eval()
with torch.no_grad():
    outputs = model(x_test_tensor)
    _, predicted = torch.max(outputs, 1)
    accuracy = accuracy_score(predicted.numpy(), y_test)

print('{}% of samples were classified correctly!'.format(accuracy * 100))

```

#### 4. Evaluation and Testing

Once trained, the model is evaluated on the test set to assess its performance:

- **Accuracy Calculation:** The model's predictions are compared to the actual labels to calculate the accuracy.
- **Confusion Matrix:** A confusion matrix can be used to visualize the classification results and identify potential misclassifications.

#### 5. Real-Time Gesture Recognition

After training, the model is integrated into a real-time application using a webcam. The system captures live video frames, extracts hand landmarks using MediaPipe, normalizes the coordinates, and feeds the data to the trained model for prediction.

#### Inference Pipeline

- **Hand Landmark Detection:** The real-time video frames are processed to extract hand landmarks.
- **Data Preparation:** The extracted landmarks are normalized and formatted into a 42-element feature vector.
- **Prediction:** The model predicts the gesture class, which is then displayed on the video frame along with a bounding box around the detected hand.

```
# Initialize video capture
cap = cv2.VideoCapture(0)

# Initialize MediaPipe hands
mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles

# Configure MediaPipe Hands
hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.3)

# Label dictionary
labels_dict = {0: "Hello", 1: "Please", 2: "Thank you", 3: "OK", 4: "Thumbs up"}

while True:
    data_aux = []
    x_ = []
    y_ = []

    # Capture frame-by-frame
    ret, frame = cap.read()
    if not ret:
        break

    H, W, _ = frame.shape
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

```

# Process the image and detect hands
results = hands.process(frame_rgb)
if results.multi_hand_landmarks:
    for hand_landmarks in results.multi_hand_landmarks:
        mp_drawing.draw_landmarks(
            frame,
            hand_landmarks,
            mp_hands.HAND_CONNECTIONS,
            mp_drawing_styles.get_default_hand_landmarks_style(),
            mp_drawing_styles.get_default_hand_connections_style()
        )

# Collect x, y coordinates of each Landmark
for landmark in hand_landmarks.landmark:
    x = landmark.x
    y = landmark.y
    x_.append(x)
    y_.append(y)

# Normalize landmarks and construct data_aux with 42 elements
for landmark in hand_landmarks.landmark:
    data_aux.append(landmark.x - min(x_))
    data_aux.append(landmark.y - min(y_))

# Ensure data_aux matches the expected input shape of the model
data_aux = data_aux[:42] # Keep only 42 values if more were added

# Bounding box coordinates for hand
x1 = int(min(x_) * W) - 10
y1 = int(min(y_) * H) - 10
x2 = int(max(x_) * W) + 10
y2 = int(max(y_) * H) + 10

```

## 6. System Performance and Challenges

- **Model Accuracy:** The model achieves a high classification accuracy on the test set, making it suitable for real-time applications.
- **Challenges:**
  - Variability in lighting conditions, background clutter, and hand positioning can affect the accuracy of the hand landmark detection.
  - The model currently focuses on static gestures, and extending it to dynamic gestures (involving motion) would require additional

modifications, such as using Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks for temporal sequence modelling.

## 7. Future Enhancements

- **Dynamic Gesture Recognition:** Incorporating LSTM layers to handle sequences of frames for dynamic gesture recognition.
- **Data Augmentation:** Applying techniques like random rotation, scaling, and flipping to increase the diversity of the training dataset.
- **Deployment:** Packaging the model into a lightweight application for

deployment on mobile devices or embedded systems.

This model demonstrates a practical implementation of deep learning techniques for real-time sign language recognition, contributing to accessibility solutions for the deaf and hard-of-hearing community.

```
# Inference
with torch.no_grad():
    inputs = torch.tensor(data_aux, dtype=torch.float32).unsqueeze(0) # Add batch dimension
    outputs = model(inputs) # Pass the input through the model
    predicted_index = torch.argmax(outputs, dim=1).item() # Get predicted class index
    predicted_character = labels_dict.get(predicted_index, "Unknown") # Get label or default to "Unknown"

# Display bounding box and prediction on frame
cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 0), 4)
cv2.putText(frame, predicted_character, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 1.3, (0, 0, 0), 3,
            cv2.LINE_AA)

# Display the resulting frame
cv2.imshow('frame', frame)
if cv2.waitKey(1) & 0xFF == ord('q'): # Exit if 'q' key is pressed
    break

# Release the capture and close windows
cap.release()
cv2.destroyAllWindows()
```

## 5. DATA EXPLORATION

Before training a machine learning model, it is crucial to thoroughly explore and understand the dataset to ensure its quality and to gain insights into its structure. In the context of this Sign Language Recognition project, data exploration involves analyzing the collected hand gesture images and extracted landmark features to identify any potential issues, optimize preprocessing steps, and prepare the data for effective training.

### 1. Overview of the Dataset

The dataset used in this project consists of hand gesture images captured for five predefined gesture classes: **"Hello", "Please", "Thank You", "OK", and "Thumbs Up"**. Each class contains 1,000 samples, resulting in a balanced dataset with a total of 5,000 images.

- **Data Distribution:** Analyze the distribution of samples across different gesture classes to ensure that the dataset is balanced. This can help prevent bias in the model during training.

- **Image Dimensions:** Check the consistency of image sizes and resolutions to ensure uniformity, which is crucial for accurate feature extraction.
- **File Format and Integrity:** Verify that all images are properly saved in the required format (e.g., .jpg) and are free from corruption.

## 2. Landmark Feature Analysis

The primary features used for training the model are derived from the 21 hand landmarks detected by MediaPipe. Each image is converted into a set of 42 features (x, y coordinates of each landmark).

- **Feature Distribution:** Visualize the distribution of the extracted coordinates to understand their ranges, identify outliers, and detect any inconsistencies.
- **Correlation Analysis:** Analyze the relationships between different landmarks to identify patterns or redundant features. This can help in feature selection or dimensionality reduction.
- **Normalization Check:** Ensure that the normalization of coordinates (relative to the minimum x and y values) is working correctly and that the resulting values are within the expected range

## 3. Data Quality and Cleaning

- **Missing or Incomplete Data:** Check for any missing values or incomplete entries in the dataset. While this is less common in image-based datasets, it is still important to confirm.
- **Noise and Variability:** Identify images with poor lighting conditions, motion blur, or other distortions that may affect the accuracy of hand landmark detection. These can be filtered or enhanced during preprocessing.
- **Class Imbalance:** Although the dataset is intentionally balanced, it is crucial to verify this before training to avoid model bias.

## 4. Visual Exploration

- **Sample Visualization:** Display sample images and their corresponding landmarks to ensure that the hand detection and landmark extraction processes are functioning correctly.
- **Landmark Distribution Plots:** Plot the coordinates of the landmarks to visualize the shape and orientation of gestures. This can reveal patterns specific to each gesture class.

## 5. Statistical Analysis

- **Descriptive Statistics:** Calculate basic statistics (mean, median, standard deviation) for the

landmark coordinates across all classes to gain insights into the variability and range of hand positions.

- **Outlier Detection:** Use statistical techniques to identify outliers in the landmark coordinates, which may indicate errors in the data collection or preprocessing pipeline.

By performing thorough data exploration, we can gain a deeper understanding of the dataset, optimize the preprocessing steps, and address any data-related challenges before training the model. This ensures that the model is trained on high-quality data, leading to better performance and generalization.

## 4. RESULTS

After training the sign language recognition model, it is essential to evaluate its performance to determine its effectiveness in accurately classifying hand gestures. The results provide insights into the model's strengths and areas for potential improvement. This section presents a detailed analysis of the model's performance on the test dataset, including key metrics, visualizations, and discussions of the results.

### 1. Model Accuracy

- **Training and Test Accuracy:** The accuracy of the model is measured on both the training set and the test

set. The test accuracy reflects the model's generalization capability to unseen data.

- **Confusion Matrix:** A confusion matrix is generated to visualize the model's performance across different gesture classes. This matrix highlights how well the model distinguishes between the five gesture categories: "Hello", "Please", "Thank You", "OK", and "Thumbs Up".

- **True Positives (TP):**  
Correctly predicted gestures.
- **False Positives (FP):**  
Incorrectly predicted gestures that were classified as a different class.
- **False Negatives (FN):**  
Actual gestures that were missed by the model.

### 2. Precision, Recall, and F1-Score

- **Precision:** Indicates the proportion of correct positive predictions among all predictions made for a class. High precision means fewer false positives.
- **Recall (Sensitivity):** Measures the model's ability to detect all instances of a specific class. High recall indicates that the model is effectively identifying gestures.
- **F1-Score:** Provides a harmonic

mean of precision and recall, especially useful when dealing with imbalanced data or when both false positives and false negatives need to be minimized.

### 3. Loss Curve Analysis

- The **training loss** and **validation loss** curves are plotted over the epochs to assess the convergence of the model. A gradual decrease in loss indicates that the model is learning effectively.
- **Overfitting Detection:** If there is a significant gap between training and validation loss towards the end of training, this could indicate overfitting. Strategies such as dropout, regularization, or early stopping may be explored to mitigate overfitting.

### 4. Real-Time Inference Results

- **Live Gesture Recognition:** The trained model is integrated into a real-time system using a webcam. The system's performance in recognizing gestures in real-world conditions (varying lighting, backgrounds, and hand positions) is evaluated.
- **Latency and Processing Speed:** The response time of the model during real-time predictions is measured to ensure that it can be deployed for practical applications

without noticeable delays.

### 5. Analysis of Misclassifications

- **Error Analysis:** By examining the confusion matrix and individual misclassified samples, patterns are identified where the model struggled (e.g., similar gestures being confused due to overlapping features).
- **Common Failure Cases:** Discussion on specific scenarios where the model performed poorly, such as variations in hand orientation, lighting conditions, or background noise.

### 6. Comparison with Existing Models

- **Benchmarking:** The model's performance is compared with other sign language recognition models from the literature, highlighting improvements in accuracy, robustness, or efficiency.
- **Ablation Study:** An analysis of the impact of different model components (e.g., the number of hidden layers, activation functions, or feature normalization) on overall performance.

### 7. Summary of Findings

The results indicate that the proposed model is effective in recognizing the selected sign language gestures, achieving a high accuracy rate on the test dataset. However, areas for improvement have

been identified, particularly in handling real-time variations and ambiguous gestures. The insights from this evaluation will guide future enhancements and optimizations of the system.

## 5. CONCLUSION

The development of an automated Sign Language Recognition (SLR) system presents significant potential to enhance communication between individuals with hearing impairments and the wider community. This project successfully implemented a real-time gesture recognition system using a combination of computer vision techniques (Mediapipe) and deep learning (PyTorch). The proposed model was trained to classify five commonly used hand gestures, achieving high accuracy and reliable performance on test data.

### Key Findings

The results demonstrated that the model could effectively detect and classify sign language gestures with high precision and accuracy. The use of hand landmarks extracted through Mediapipe, combined with a simple feedforward neural network, proved to be a robust approach for static hand gesture recognition. The model achieved strong performance metrics, indicating its potential for real-world applications.

### Challenges and Limitations

Despite the success of the system, several challenges were identified during the project:

- **Variability in Real-World Conditions:** The model's performance could be affected by changes in lighting, background clutter, and hand positioning, which were not fully addressed during training.
- **Static Gestures Only:** The current system is limited to recognizing static gestures. The extension to dynamic gestures, which involve movement sequences, remains an area for future research.

### Future Work

Building on the achievements of this project, several improvements and extensions can be pursued:

- **Dynamic Gesture Recognition:** Incorporating Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks to handle temporal sequences for recognizing dynamic gestures.
- **Data Augmentation:** Expanding the dataset with additional variations in hand orientations, lighting, and backgrounds to enhance robustness.
- **Mobile and Embedded Deployment:** Optimizing the model

for deployment on mobile devices or embedded systems, making it accessible for use in real-world settings.

### Concluding Remarks

This project demonstrates the feasibility of using computer vision and deep learning techniques for real-time sign language recognition. By leveraging accessible technologies like webcams and open-source frameworks, the system provides a cost-effective solution that can be further developed to support communication for the deaf and hard-of-hearing community. The continued refinement of this technology holds promise for creating inclusive communication tools that can significantly impact the lives of individuals relying on sign language.

## 6. REFERENCES

1. A. K. Sahoo, G. S. Mishra, and K. K. Ravulakollu, "Sign Language Recognition: State of the Art," *ARPJ Journal of Engineering and Applied Sciences*, vol. 9, no. 2, pp. 116-120, Feb. 2014. [Online]. Available: [https://www.researchgate.net/publication/262187093\\_Sign\\_language\\_recognition\\_State\\_of\\_the\\_art](https://www.researchgate.net/publication/262187093_Sign_language_recognition_State_of_the_art)
2. I. A. Adeyanju, O. O. Bello, and M. A. Adegboye, "Machine Learning Methods for Sign Language Recognition: A Critical Review and Analysis," *Intelligent Systems with Applications*, vol. 12, p. 200056, Dec. 2021. [Online]. Available: <https://doi.org/10.1016/j.iswa.2021.200056>
3. R. K. Pathan, M. Biswas, S. Yasmin, M. U. Khandaker, M. Salman, and A. A. F. Youssef, "Sign Language Recognition Using the Fusion of Image and Hand Landmarks through Multi-Headed Convolutional Neural Network," *Scientific Reports*, vol. 13, no. 16975, Sep. 2023. [Online]. Available: <https://doi.org/10.1038/s41598-023-43852-x>