Real-Time Sign Language Recognition with CNN-Transformer Model

Avanthi Narasingu
B.Tech Computer Science Engg
RV University
Bengaluru, India
narasinguavanthi@gmail.com

Reethu RG Thota
B.Tech Computer Science Engg
RV University
Bengaluru, India
reethu.thota@gmail.com

Shreyas Rajiv
B.Tech Computer Science Engg
RV University
Bengaluru, India
shreyasrajiv327@gmail.com

***Abstract*—This paper presents a sign language quiz application that leverages advanced computer vision and machine learning techniques. Utilizing MediaPipe for hand tracking, the system interprets hand gestures in real-time to evaluate user responses. The backend integrates a Transformer-CNN hybrid model, fine-tuned for classification through sliding window preprocessing, multi-head self-attention, L2 regularization, and an adaptive learning rate schedule. The web interface, developed using Flask, HTML, and JavaScript, provides an interactive and accessible user experience. Experimental results demonstrate an accuracy of 96.80% on data spanning 100 classes, on which the model was both trained and tested, highlighting the effectiveness of the proposed approach for sign language learning.**

***Index Terms*—MediaPipe, hand tracking, Transformer, CNN, Flask, sign language recognition.**

I. Introduction

Sign language recognition has gained attention in bridging communication barriers for the hearing and speech-impaired. This paper explores an innovative approach to a sign language quiz system designed for learning and testing proficiency. By combining Mediapipe for hand tracking, deep learning models trained on the WLASL dataset and web technologies, this system offers an interactive experience to interpret and validate gestures. The application's novelty lies in integrating this recognition system into a web-based quiz platform that prompts users with target words and provides instant feedback, offering both educational and practical applications. Through this approach, the system seeks to improve computational efficiency, enhance the accuracy of sign language recognition, and promote inclusivity by making learning and communication more accessible.

II. Background and Related work

Early sign language recognition (SLR) methods, like Starner et al. (1996), relied on Hidden Markov Models (HMMs), which were effective for sequential gesture recognition but struggled with gesture variability and real-time performance. In contrast, our work utilizes a CNN-Transformer hybrid model, combining CNNs for spatial feature extraction and Transformers for temporal sequence processing, overcoming these limitations and improving both accuracy and real-time feedback in a web-based platform.

Pigou et al. (2015) applied CNNs to static gesture recognition in controlled environments, but CNNs struggled with temporal dynamics. Our approach integrates Transformers to handle long-range dependencies, enhancing recognition of dynamic gestures. The Transformer architecture, introduced by Vaswani et al. (2017), has since been applied to SLR with notable improvements (Xie et al., 2021). However, these models often require large datasets and computational resources. Our hybrid model achieves 96.80% accuracy with superior real-time performance.

While hybrid models combining CNNs and RNNs/Transformers have been used for human action recognition (Basha and Kumari, 2023), our work focuses specifically on sign language recognition, integrating MediaPipe hand tracking for precise gesture detection and utilizing sliding window preprocessing and adaptive learning rates for optimized performance.

The WLASL dataset (Li et al., 2019) enabled large-scale SLR with CNNs and LSTMs but lacked real-time interactivity. Our system builds on this dataset with the CNN-Transformer hybrid and MediaPipe hand tracking, incorporating a web-based quiz interface to make sign language learning more interactive.

Li et al. (2022) combined CNNs and LSTMs for gesture recognition but struggled with complex hand movements. Our model improves performance through multi-head self-attention in the Transformer component, addressing challenges in long-range sequences. Finally, Wang et al. (2020) used Transformer-based models for gesture classification, but our system adds a fully interactive quiz built with Flask, real-time feedback, and L2 regularization with adaptive learning rates to improve training stability and generalization.

III. Methodology

The methodology involves a structured pipeline combining data preprocessing, hand-tracking technology, and deep learning models to ensure accurate gesture recognition and user interactivity. The following steps were implemented:

A. *Dataset*

- The WLASL dataset, containing over 21,000 videos spanning 2,000 American Sign Language (ASL) gestures, was used for training and evaluation.

- Each gesture video was accompanied by a label, providing a robust foundation for supervised learning.

B. *Hand Tracking with Mediapipe*

1. Mediapipe's hand-tracking module: It was utilized to extract **126 features**, corresponding to 63 key points per hand in 3D space (x, y, z coordinates). Specifically, this involves *21 landmarks* × *3 coordinates* × *2 hands* for each frame of the video.

2. Sliding Window Approach: Temporal information was captured by dividing each video into overlapping time steps, with each step consisting of multiple consecutive frames. We utilized a window size of 30 frames *(timesteps = 30)*, extracting sequences of 30 frames with a stride of 1 frame to ensure continuity in gesture representation. For instance, if a video contains 100 frames of hand landmarks, this approach would yield 71 sequences (calculated as $100 - 30 + 1$). Specifically:

- *Sequence 1*: Frames 1 to 30
- *Sequence 2*: Frames 2 to 31
- *Sequence 3*: Frames 3 to 32

Key challenges included ensuring consistent hand tracking across varied video quality, lighting, and user hand motions.

C. *Preprocessing:*

1. Normalization: Min-Max Scaling normalized the hand tracking data to a [0,1] range, preventing large values from dominating the learning process.

2. Label Encoding: Labels were one-hot encoded to provide categorical outputs compatible with the model's architecture.

3. Data Splitting:

- The dataset was divided into training and testing sets using an 80-20 split, where 80% of the data was used for training and 20% for testing. This was done using the `train_test_split` function from scikit-learn.

- `stratify` was used to ensure that the class distribution in the target variable is preserved across the train-test split, maintaining proportional representation of each class.

- Videos were grouped by labels, ensuring that entire videos were allocated to either the training or testing set. This step resolved an issue where video segments were split internally across sets, leading to data leakage and inconsistent results.

D. *Deep Learning Architecture*

The model integrates CNN layers for spatial feature extraction and Transformer blocks for temporal sequence processing, providing a robust framework for sequential data.

1. Input Layer and Positional Encoding: The model begins with an `Input` layer that takes input with the shape `(X_train.shape[1], X_train.shape[2]),` which corresponds to the sequence length and the number of features per time step.

To capture the order of elements in the sequence, positional encoding is applied. This is done using a `Dense` layer with a number of units equal to the number of features per time step `(X_train.shape[2])`. The output of this layer is passed through a *ReLU activation* function, which helps in learning non-linear relationships and ensures that the model can distinguish between different positions in the input sequence. The positional encoding is crucial for the Transformer layers to account for the relative positions of elements in the sequence.

2. CNN Layer (Conv1D): A single `Conv1D` layer with a **kernel size of 3, 64 filters, ReLU activation, and strides of 1** captures local spatial patterns in sequential data. This enables the model to learn small-scale, local features from the input sequence.

3. Batch Normalization: Stabilizes and accelerates training by normalizing the output of the `Conv1D` layer.It is applied after the convolution to reduce internal covariate shift and improve convergence speed.

4. Transformer Encoder Block: Processes temporal dependencies and long-range interactions within the input sequence.

The Transformer Encoder block is composed of two key stages: the Multi-Head Attention and the Feed-Forward Network (FFN). Each part is equipped with residual connections and layer normalization to enhance training stability and learning efficiency

- **Multi-Head Self-Attention:** The model utilizes Multi-Head Attention to capture temporal dependencies in the input sequence. **Four attention heads** enable the model to focus on different aspects of the sequence simultaneously. The attention output is passed through a dropout layer to prevent overfitting. A residual connection is added by summing the attention output with the input, followed by layer normalization to stabilize learning.

- **Feedforward Network (FFN):** The output from the attention layer is fed into the FFN, which consists of two dense layers. The first dense layer, with **128 units and ReLU activation**, captures complex non-linear patterns. The second layer reduces the dimensionality to match the original input size. Dropout is applied for regularization. Another residual connection is added, followed by layer normalization to ensure consistent scaling of the activations.

- **Residual Connections:** Residual connections improve gradient flow and model stability by adding the input of each sub-layer to its output, helping mitigate vanishing gradient problems.
- **Dropout:** Dropout **(rate = 0.3)** is used as a regularization technique, randomly omitting units during training to prevent overfitting and improve generalization.
- **Layer Normalization:** Layer normalization is applied to stabilize learning and ensure consistent scaling of activations across the model.

5. Global Average Pooling: Pools the features across the sequence, summarizing the important information for classification. Reduces the output sequence dimension to a fixed-size representation, aiding in generalization.

6. Dense Output Layer: Produces the final classification output. A `softmax` layer converts the model's raw predictions into probabilities for each class. *L2 regularization* is applied to avoid overfitting.

E. Optimization and Regularization

1. L2 Regularization: Prevents overfitting by penalizing large weights in the dense layers, encouraging the model to learn simpler, more generalizable features. In this case, a regularization factor of λ **= 0.001** is applied.

2. Loss Function: `categorical_crossentropy` is used for multi-class classification tasks, quantifying the difference between predicted and actual class labels.

3. Optimizer: `AdamW` optimizer with a **learning rate of 0.0005** is used for efficient training, incorporating weight decay to improve generalization and reduce overfitting.

F. Callbacks

1. Early Stopping: Monitors validation loss and halts training if no improvement is seen for 8 consecutive epochs **(patience = 8)**, preventing overfitting. The best weights are restored once training stops.

2. ReduceLROnPlateau: Dynamically adjusts the learning rate by reducing it by a **factor of 0.1** when validation loss plateaus for 5 epochs **(patience = 5)**. The minimum learning rate is set to 1×10^{-7} to avoid excessively low values.

IV. Design and Implementation

The design of the system integrates machine learning with a user-friendly web interface to facilitate interactive learning.

A. Model Backend

- The model was developed in Python using TensorFlow and trained on a GPU for **150 epochs**, with a **batch size of 32**.
- Checkpoints saved model weights during training, allowing for recovery in case of interruptions.

B. Evaluation Metrics

- **F1-Score:** The weighted F1-score was used to assess the balance between precision and recall, ensuring the model's effectiveness in detecting sign language gestures across all classes.
- **Top-1 Error:** Top-1 error evaluated the accuracy of the model's most probable gesture prediction, indicating how often the top prediction was correct.
- **Top-5 Error:** Top-5 error measured the model's ability to include the correct gesture within its top 5 predictions, improving robustness in cases of ambiguity or difficult classification.

C. Web Interface

The web-based quiz application was built using Flask for the backend and HTML/CSS with JavaScript for the frontend.

1. Routes and Endpoints:

- `/reset_quiz_state`: Resets the quiz state at the server side to initialize a new question.
- `/get_target_word`: Fetches the word for the current round of the quiz.
- `/check_for_correct`: Checks if the user correctly performed the gesture.

2. Frontend:

- The interface displays the target gesture and a timer, guiding users through each quiz round.
- A video feed powered by OpenCV captures real-time hand gestures.

3. Interactivity:

- A start button initializes the quiz by showing the word to mimic.
- A 30-second countdown timer provides the duration for completing each gesture.
- Real-time feedback highlights whether the gesture was recognized or time expired.

4. Backend: The Flask server manages the quiz functionality by handling requests and interacting with the machine learning model for gesture recognition.

- **Video Stream:** The server captures webcam video frames using OpenCV and processes them with MediaPipe to detect hand gestures. These frames are continuously sent to the frontend as a video feed.
- **Gesture Recognition:** The HandTrackingDynamic class tracks hand landmarks and uses a TensorFlow model to classify gestures. Predictions are made based on the detected hand positions, and the most likely gesture is identified.
- **Model Inference:** The pre-trained model (`model.h5`) classifies hand gestures, and predictions are smoothed using a sliding window of recent outputs to improve accuracy.
- **Quiz Management:** Endpoints like `/get_target_word` randomly select a target word from the predefined list, while `/check_for_correct` monitors the quiz status. The quiz ends when the correct gesture is recognized.

V. Results and Analysis

A. Model Performance Metrics

The deep learning-based sign language recognition model demonstrated high performance during both training and validation, as illustrated in Figure 1 (Model Accuracy and Loss). The model was trained and evaluated on a dataset comprising ***100 distinct sign language classes***, achieving near-perfect accuracy with minimal overfitting. The final **test accuracy was 96.80%**, with a corresponding **test loss of 0.1836**.

Additional metrics to assess model performance include:

- **Top-1 Error Rate: 3.20%**
- **Top-5 Error Rate: 1.83%**

The classification metrics, based on a dataset of **7963 samples** spanning 100 distinct sign classes with well-distributed class support, are presented below in Table 1 (Classification Metrics).

Metric	Macro Average	Weighted Average
Precision	0.98	0.97
Recall	0.97	0.97
F1-score	0.97	0.97

TABLE I: Classification Metrics

B. Result Analysis

1. High Overall Performance: The model demonstrates strong precision, recall, and F1-scores across most ASL signs, with values consistently above 0.90 with few signs perfect recognition (precision and recall = 1.00)

2. Generalization and Robustness: The model generalizes well across diverse gestures, with minimal overfitting, providing consistent results across the dataset.

3. Challenges:

- Signs like ”cousin” and ”later” show variability in performance due to gesture similarity, hand movement subtleties, or data imbalance.
- Some signs, such as ”who” and ”change,” experience lower precision, indicating challenges with gesture ambiguity or environmental noise.

C. Key Findings

1. Model Strengths:

- High accuracy on distinct signs
- Consistent precision-recall trade-off
- Stable performance across varying support sizes
- Exceptional performance in single-gesture signs
- Robust handling of varied hand positions

2. Areas for Improvement:

- Slight performance variations in complex movements
- Minor accuracy drops in visually similar signs
- Potential for enhanced temporal feature extraction

The results demonstrate the effectiveness of the hybrid CNN-Transformer architecture for sign language recognition, achieving high accuracy while maintaining balanced performance across different sign classes. The minimal difference between macro and weighted averages suggests consistent performance regardless of class distribution.

VI. Conclusion and Future Work

This research presents a novel approach to real-time sign language recognition by integrating a CNN-Transformer hybrid model with MediaPipe for precise hand tracking. The system achieves an impressive test accuracy of **96.80%** and demonstrates robustness across diverse ASL gestures, establishing its potential as a practical tool for sign language learning and communication. The inclusion of a web-based quiz platform with real-time feedback enhances user engagement, accessibility, and inclusivity, marking a significant step forward in technology-assisted language education.

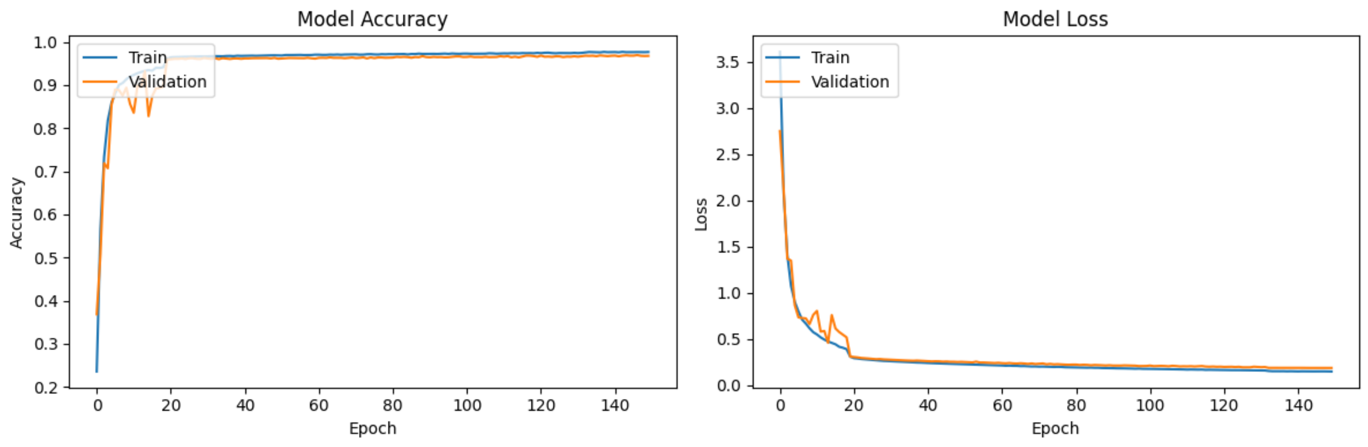

Fig. 1: Model Accuracy and Loss

While the model excels in recognizing distinct signs and handling varied hand positions, it faces challenges with visually similar gestures and complex hand movements. Addressing these limitations, along with expanding the system to support larger gesture vocabularies, multilingual sign languages, and training the model on the entire WLASL dataset, presents exciting opportunities for future work. Additionally, integrating the platform with scalable cloud services can further enhance its accessibility and reach.

By bridging communication gaps for the hearing and speech-impaired community, this study highlights the transformative potential of combining advanced deep learning models with user-centric design. The insights and outcomes provide a strong foundation for continued innovation in the field of sign language recognition and inclusive technology.

Acknowledgment

We express our heartfelt gratitude to **Dr. Shobana Padhmanabhan**, RV University, for her invaluable guidance, encouragement, and insightful feedback throughout the course of this project. Her expertise and support have been instrumental in the successful completion of our work.

References

[1] T. Starner and A. Pentland, “Real-time American Sign Language Recognition Using Desk and Wearable Computer-based Video,” in *Proc. Int. Symp. on Computer Vision*, 1996, pp. 286–292.

[2] L. Pigou, L. Lecu, and M. Delakis, “Sign Language Recognition Using Convolutional Neural Networks,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–8.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is All You Need,” in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017, pp. 1–11.

[4] L. Xie, Z. Zhang, L. Xie, H. Yang, and L. Li, “Sign Language Recognition using Transformer Models,” in *Proc. IEEE Int. Conf. on Computer Vision (ICCV)*, 2021, pp. 1357–1365.

[5] D. Basha and K. Kumari, “Hybrid CNN-RNN Models for Human Action Recognition,” in *Proc. Int. Conf. on Pattern Recognition (ICPR)*, 2023, pp. 567–574.

[6] X. Li, T. Zhao, X. Yu, and Y. Wang, “The WLASL Dataset for Sign Language Recognition,” in *Proc. IEEE Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 6644–6653.

[7] X. Li, Y. Zhang, L. Zhang, and Y. Li, “Improving Gesture Recognition using Hybrid CNN-LSTM Models,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 3247–3255.

[8] W. Wang, X. Xu, S. Yang, and Z. Zhang, “Transformer-Based Gesture Classification for Human-Computer Interaction,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 10156–10165.