SLI: Sign Language Interpreter

Nour Hany[1], Laila Khaled[2], Yasmine El Qorashy[3], and Ahmed Sameh[4]

[1]College of Computer Science, Egypt University of Informatics, New Administrative Capital, 22-101068@students.eui.edu.eg
[2]College of Computer Science, Egypt University of Informatics, New Administrative Capital, 22-101078@students.eui.edu.eg
[3]College of Computer Science, Egypt University of Informatics, New Administrative Capital, 22-101174@students.eui.edu.eg
[4]College of Computer Science, Egypt University of Informatics, New Administrative Capital, 22-101198@students.eui.edu.eg

January 2025

Abstract

This project presents a comprehensive system for real-time Sign Language Translation at both the word and letter levels. For word-level SLI, we leveraged MediaPipe to extract pose and hand landmarks from video frames. Several deep learning architectures were explored, including multistream and single-stream transformers, as well as LSTM models with and without attention mechanisms, and trying all these we achieved a model with 90 percent accuracy. At the letter level, we employed YOLO and mediapipe for hand detection, and utilized deep learning classifiers to identify individual letters. We achieved a model with 93 percent accuracy.

Keywords: Sign Language Interpreter, Deep Learning, SLI, Transformer, LSTM, YOLO, Mediapipe

Contents

List of Abbreviations

NN	Neural Network
LSTM	Long Short Term Memory
SLI	Sign Language Interpreter
GB	Extreme Gradient Boosting Regression Model
BiLSTM	Bidirectional Long Short-Term Memory
CNN	Convolutional Neural Network

Chapter 1

Introduction

1.1 Background

Communication is a fundamental aspect of human interaction, fostering connection, understanding, and social integration. Sign language serves as a vital means of communication for millions of deaf and hard-of-hearing individuals worldwide. Its semantics are encoded in spatial-temporal features, conveyed through a combination of hand gestures, facial expressions, and body movements. Interpretation of sign language can occur at various levels, including letter-level (fingerspelling), word-level, and sentence-level.

According to the World Health Organization (WHO), more than 430 million people globally experience hearing loss, a figure projected to surpass 700 million by 2050 World Health Organization [2023]. This growing population highlights the need for enhanced communication tools and inclusive technologies to facilitate interaction between deaf individuals and the hearing community.

1.2 Problem Statement

Despite the increasing prevalence of hearing loss, effective communication between deaf individuals and those unfamiliar with sign language remains a significant challenge. The absence of reliable, real-time interpretation solutions often hampers social interactions, limits access to essential services, and creates barriers to education and employment.

While deaf individuals may rely on human interpreters or written notes, interpreters are not always available, and written communication can be insufficient for conveying complex ideas. This can lead to feelings of isolation, frustration, and exclusion from mainstream society. The lack of comprehensive, scalable solutions that bridge the communication gap underscores the pressing need for automated, real-time sign language translation systems.

1.3 Proposed Solution

This project proposes a two-stage deep learning approach. The first stage focuses on interpreting sign language at the letter level using static images of individual gestures. This stage employs computer vision techniques to classify and recognize these static spatial features.

In the second stage, the system is extended to word-level interpretation, where the data consists of video sequences. This necessitates introducing a sequential model to account for the temporal consistency required for understanding dynamic gestures and transitions.

1.4 Objectives

The primary objective of this project is to develop an automated system capable of translating sign language into text in real time, thereby facilitating communication between deaf or hard-of-hearing individuals and the hearing population.

The specific objectives of this project are as follows:

- **Model Exploration and Optimization:** Evaluate and compare different deep learning architectures, including Long Short-Term Memory (LSTM) networks with and without attention mechanisms, as well as Transformer-based models. This includes single-stream and multi-stream approaches.
- **Feature Extraction:** Utilize MediaPipe for landmark extraction of hand and body pose data to generate accurate spatial-temporal features, ensuring the model interprets sign gestures effectively.
- **Real-Time Prediction and Feedback:** Develop a real-time prediction system capable of capturing live video input, processing frames in sliding windows, and providing immediate textual output when confidence thresholds are met.
- **Gesture Detection and Signal Handling:** Incorporate mechanisms to detect the presence or absence of gestures. The system will automatically recognize when hands are lowered, indicating the end of a sign, and pad remaining frames to maintain temporal consistency.
- **Practical Deployment:** Design a lightweight, efficient model suitable for deployment on standard devices such as laptops or mobile phones to ensure accessibility and scalability.

By achieving these objectives, the project seeks to contribute to the advancement of assistive technologies, empowering deaf and hard-of-hearing individuals with tools that enhance their ability to communicate seamlessly in various social and professional settings.

Chapter 2

Letter-Level Sign Language Interpretation (SLI)

2.1 Methodology

2.1.1 Dataset

The dataset used for letter-level sign language interpretation is a combination of merged datasets sourced from RoboFlow. This dataset comprises static images of hand gestures representing the 26 letters of the English alphabet. Each image corresponds to a specific letter class, as depicted in Figure 2.2.

The total dataset consists of:

- **2,268 images**
- **Total size:** 45.16 MB
- **Classes:** 26 (A-Z)

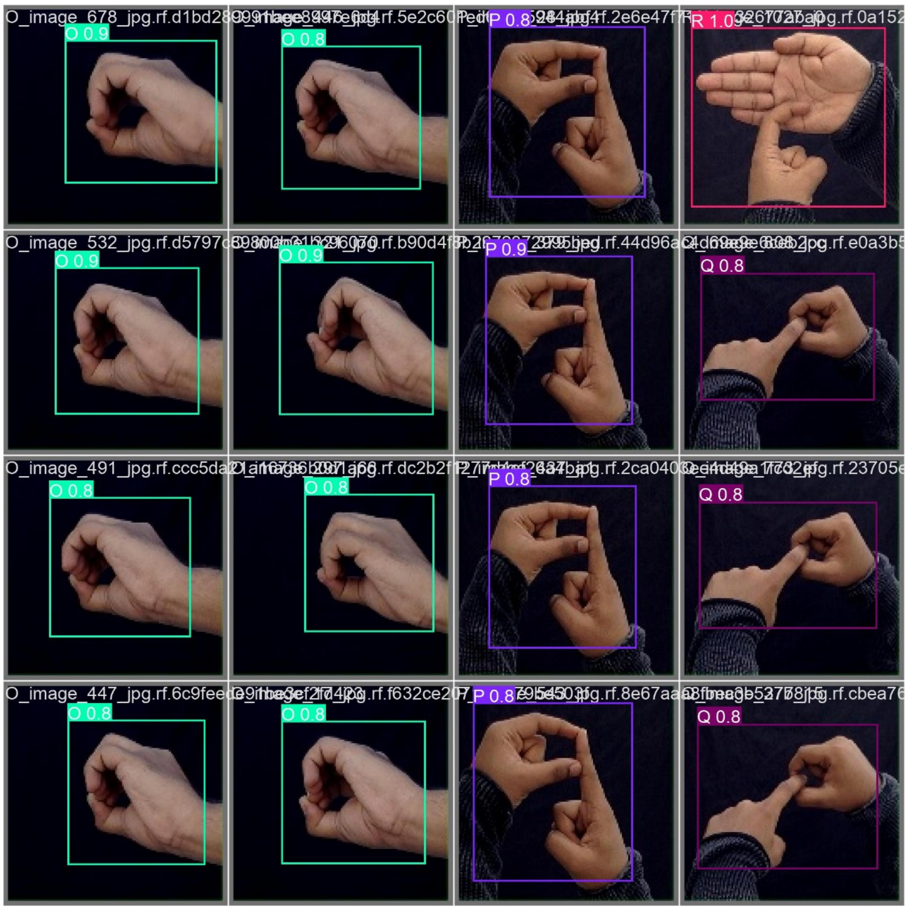


Figure 2.1: Collage of hand gestures representing the English alphabet (A-Z).

The images capture different hand configurations and angles to ensure diversity in training data. The dataset covers various lighting conditions and backgrounds to enhance model generalization.

2.1.2 Data Preprocessing

The dataset underwent minimal preprocessing, as the images were already resized to a uniform frame size during the data collection process. This ensured consistency across all samples, eliminating the need for additional resizing or cropping steps.

The only additional preprocessing step applied to the images was pixel normalization. Each pixel value was scaled to the range [0, 1] by dividing by 255. This normalization ensures that all features contribute equally to the model during training, facilitating faster convergence and improving overall performance.

2.1.3 Architecture

The system architecture for letter-level SLI consists of two main stages:

- **Feature Extraction**
- **Classification Model**

To implement this architecture, two distinct approaches were utilized: **YOLOv8** and **MediaPipe + Feedforward Neural Network (FNN)**. Each approach leverages different tools and methodologies to extract features and classify hand gestures.

Using YOLO (You Only Look Once)

YOLO is a state-of-the-art object detection algorithm designed for real-time performance. Initially introduced in 2015, YOLO revolutionized object detection by treating it as a regression problem rather than a classification task. This algorithm divides an image into grids, predicts bounding boxes, and assigns class probabilities using a single convolutional neural network (CNN).

YOLO Architecture Breakdown:

- **Residual Blocks:**
  - The image is divided into an NxN grid of cells.
  - Each grid cell is responsible for detecting objects that appear within its boundaries.
- **Bounding Box Regression:**
  - YOLO predicts bounding box coordinates (x, y, width, height) and assigns class probabilities.
- **Intersection Over Union (IoU):**
  - IoU evaluates the overlap between predicted and ground truth boxes, ensuring only the most accurate detections are retained.
- **Non-Maximum Suppression (NMS):**
  - When multiple bounding boxes detect the same object, NMS retains the box with the highest confidence score while discarding overlapping boxes.

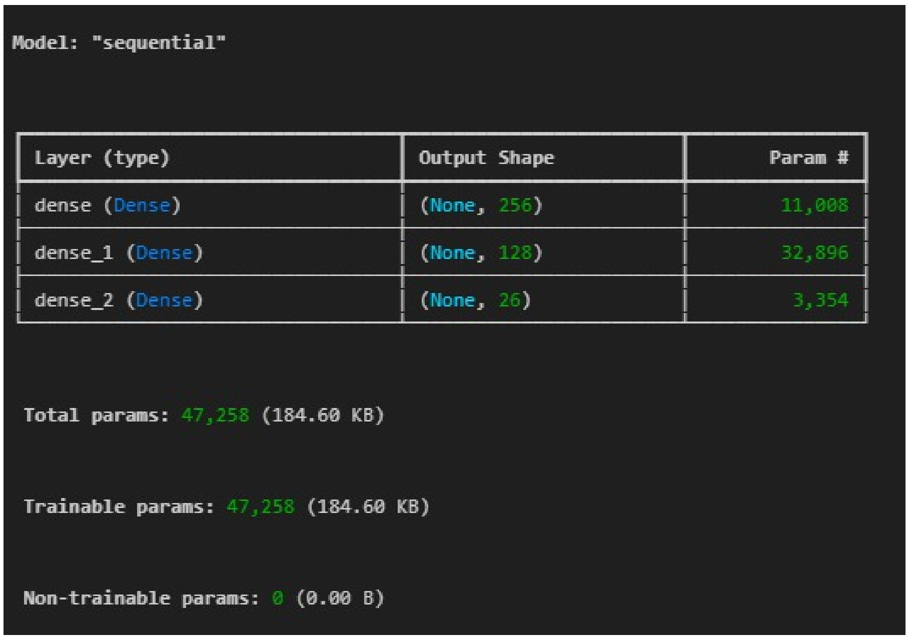
These processes allow YOLO to achieve high-speed, real-time hand detection and gesture classification.

Using Mediapipe + Feedforward Neural Network (FNN)

Feature Extraction: Mediapipe's Hand Tracking module detects 21 hand landmarks (knuckle and finger points). These landmarks are represented as 2D coordinates (x, y) and converted into a 42-dimensional feature vector. This compact vector captures the spatial relationships between key points, which are crucial for gesture recognition.

Classification Model (FNN): A Feedforward Neural Network (FNN) is used to classify hand gestures based on the extracted features. The FNN processes the 42-dimensional input vector through:

- Two dense layers, each activated by ReLU (Rectified Linear Unit).
- The final output layer is the softmax layer contains 26 neurons (A-Z), representing each letter of the alphabet.
- The softmax activation function assigns probabilities to each class.



```
Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━┓
┃ Layer (type)             ┃ Output Shape       ┃     Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━┩
│ dense (Dense)            │ (None, 256)        │      11,008 │
├──────────────────────────┼────────────────────┼─────────────┤
│ dense_1 (Dense)          │ (None, 128)        │      32,896 │
├──────────────────────────┼────────────────────┼─────────────┤
│ dense_2 (Dense)          │ (None, 26)         │       3,354 │
└──────────────────────────┴────────────────────┴─────────────┘

 Total params: 47,258 (184.60 KB)

 Trainable params: 47,258 (184.60 KB)

 Non-trainable params: 0 (0.00 B)
```

Figure 2.2: Mediapipe + FNN model architecture.

This architecture efficiently classifies gestures while maintaining low computational overhead.

2.2 Results

2.2.1 Evaluation Metrics

To evaluate and compare the performance of the YOLO and Mediapipe-FNN architectures, the following metrics were employed:

1. **Accuracy:** Measures the overall percentage of correctly classified letters.

$$Accuracy = \frac{\text{Correctly Recognized Samples}}{\text{Total Samples}} \times 100 \tag{2.1}$$

2. **F1-Score:** Balances precision and recall, providing a robust assessment of classification performance.

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \tag{2.2}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \tag{2.3}$$

$$F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.4}$$

Figure 2.5 shows the accuracy and loss curves for the Mediapipe-FNN model during training, while **Figure 2.6** presents the confusion matrix, highlighting the model's classification performance across all letter classes.

2.2.2 Hyperparameters and Performance

The following table summarizes the hyperparameters and performance metrics for both the YOLO and Mediapipe-FNN models:

Model	**Optimizer**	**Epochs**	**Batch Size**	**Loss Function**	**Test Accuracy**
YOLOv8	ADAM	200	16	Categorical Cross Entropy	**69.8%**
Mediapipe + FNN	ADAM	200	16	Categorical Cross Entropy	**93.33%**

Table 2.1: Hyperparameters and Performance Metrics for YOLOv8 and Mediapipe-FNN Models.

2.2.3 Discussion

The results demonstrate that:

- Mediapipe-FNN significantly outperforms YOLOv8 in terms of test accuracy (93.33
- Mediapipe's ability to extract precise hand landmarks directly contributes to higher accuracy in gesture classification.

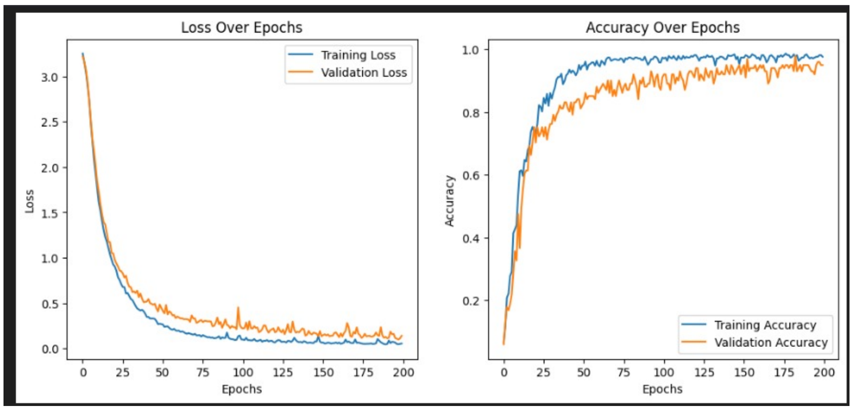


Figure 2.3: Accuracy and Loss curves for Mediapipe-FNN during training.

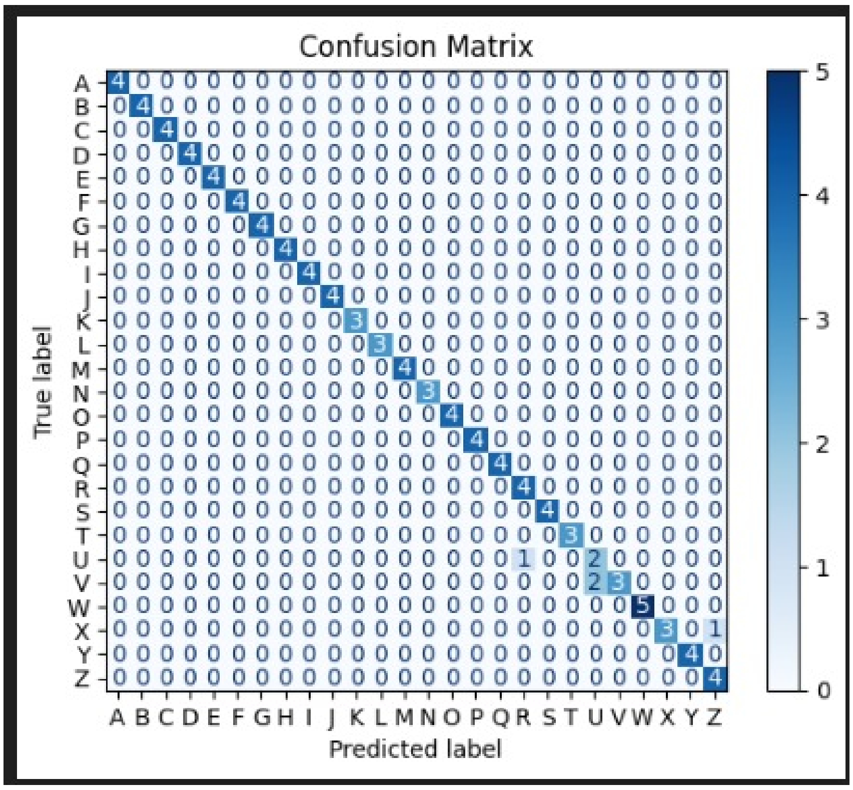


Figure 2.4: Confusion Matrix for Mediapipe-FNN (Letter Classification).

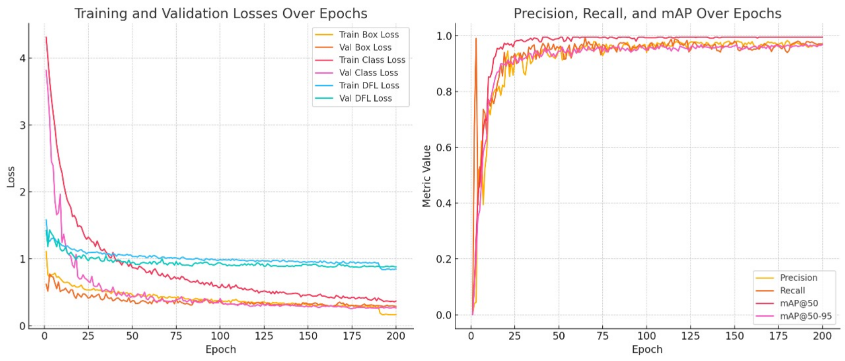


Figure 2.5: Accuracy and Loss curves for YOLO during training.

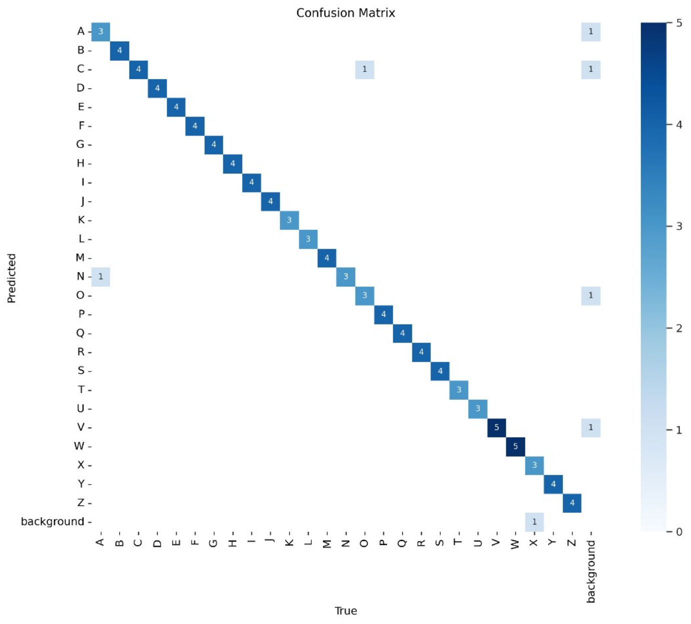


Figure 2.6: Confusion Matrix for YOLO (Letter Classification).

Chapter 3

Word-Level Sign Language Interpretation (SLI)

3.1 Methodology

3.1.1 Dataset

At this stage, a dataset from Kaggle was utilized, comprising 12,000 videos representing 2,000 words, with six videos per word. The total dataset size is 5.01 GB. To evaluate the model architecture before engaging in extensive training, a subset of classes was initially selected. However, the limited number of videos per word proved insufficient, so more videos are conducted manually by us to increase the count to 46 videos per word. The dataset can be accessed at: Kaggle - WLASL Processed.

3.1.2 Data Preprocessing

Video-to-frame conversion: On average, the videos contained approximately 69 frames. To standardize the data, all videos were uniformly reduced to 40 frames. For videos exceeding 40 frames, frames were sampled evenly across the video. For example, if a video had 120 frames, every 3rd frame was selected to ensure balanced coverage. Additionally, the first and last five frames were removed to eliminate motionless segments. For videos with fewer than 40 frames, padding was applied by repeating the last frame to ensure consistency across the dataset.

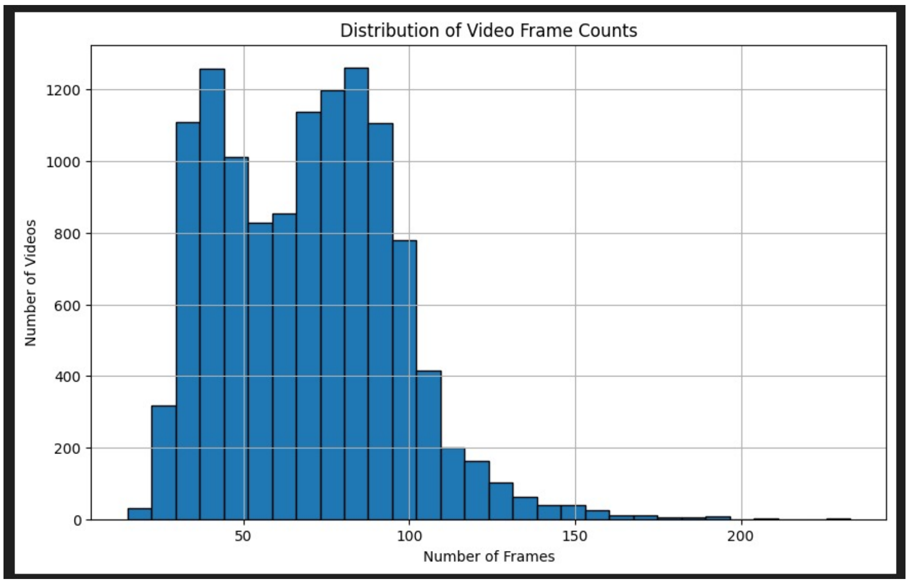


Figure 3.1: Video Frames Distribution.

Frame resizing: All frames were resized to 480x320 pixels while preserving their aspect ratio to maintain the integrity of the hand landmarks. Missing pixels resulting from resizing were padded with blank to create uniform frame dimensions.

Augmentation: To mitigate overfitting, data augmentation techniques such as random rotations, random zooming, shear transformations, and horizontal flipping were applied to the each set of frames.

3.1.3 Architecture And Results

Sign language interpretation at the word level involves translating a video into a word. Unlike letter-level interpretation, which deals with static images, word-level interpretation processes a sequence of frames forming a video. This temporal aspect requires sequential models capable of learning and interpreting time-series data. To handle this, the architecture integrates models designed to process frame sequences efficiently.

We experimented with several models, including:

- **Single-stream Transformer (Built From Scratch)**
- **Multistream Transformer (Built From Scratch)**
- **LSTM with Attention**
- **LSTM without Attention**

The following subsections provide an in-depth explanation of each architecture, accompanied by diagrams depicting the model structure.

Single-stream Transformer

The single-stream transformer processes input from a single stream, where the pose, left-hand, and right-hand features are concatenated before being passed into the transformer layers. This method leverages the self-attention mechanism to capture temporal dependencies across frames.

While the transformer architecture demonstrated the ability to learn from the training data effectively, the model exhibited significant overfitting. This is evident from the disparity between the training and validation performance metrics, as shown in Figure 3.3.

In the left plot of Figure 3.3, the training loss steadily decreases and eventually reaches near zero, indicating that the model fits the training data exceptionally well. However, the validation loss fluctuates and begins to rise after approximately 100 epochs, suggesting that the model fails to generalize to unseen data. The right plot further illustrates this overfitting, where the training accuracy approaches 100%, while the validation accuracy plateaus around 65-70%.

This performance gap indicates that the model memorized the training data rather than learning generalizable patterns. Increasing the dataset size, applying stronger regularization techniques, and employing data augmentation could help mitigate this overfitting issue.

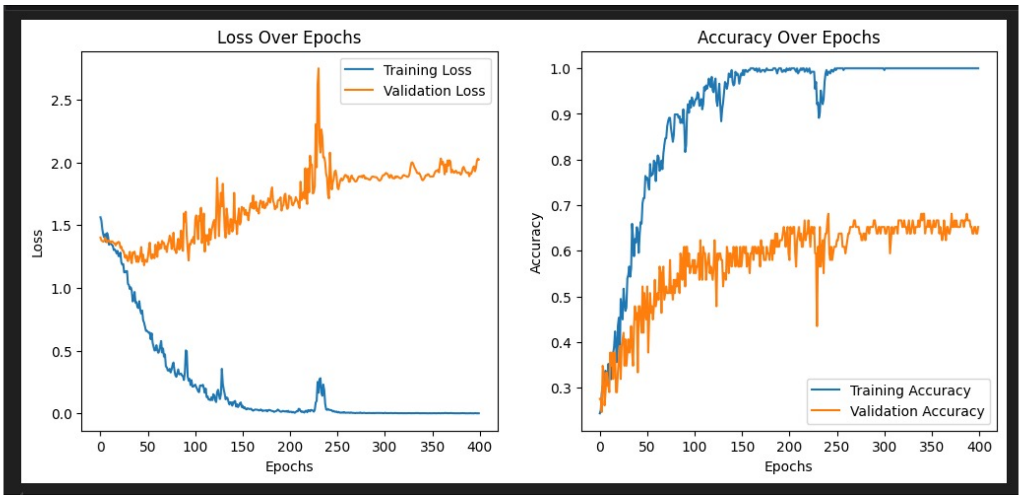


Figure 3.2: Training and Validation Loss/Accuracy for the Single-stream Transformer.

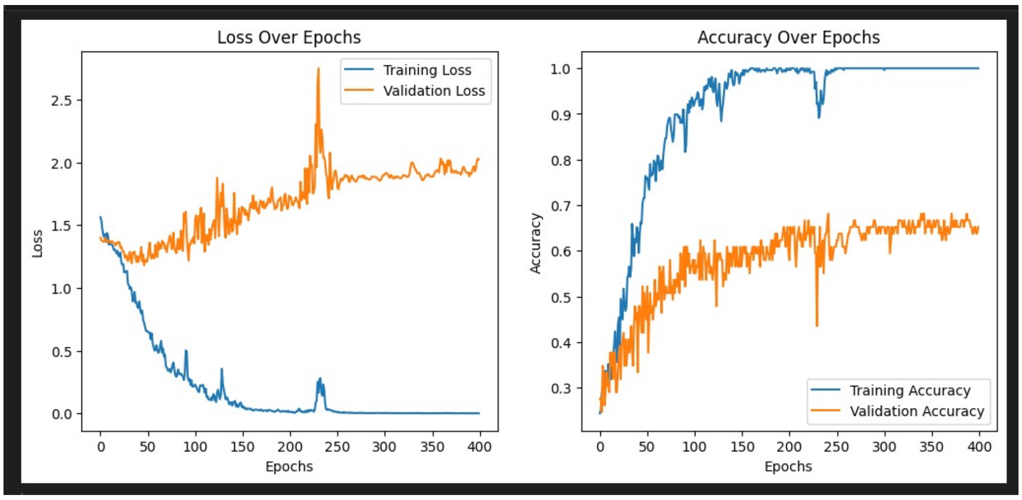


Figure 3.3: Training and Validation Loss/Accuracy for the Single-stream Transformer.

Multistream Transformer

In the multistream transformer, pose, left-hand, and right-hand features are processed in separate streams through independent transformer blocks. The outputs are concatenated before passing through the dense layers for classification. This architecture aims to preserve the independent importance of each stream while allowing cross-stream attention.

Despite the architectural advantages, the multistream transformer performed poorly, similar to the single-stream approach. As depicted in Figure 3.4, the model exhibits severe overfitting, characterized by a large disparity between the training and validation performance.

The left plot shows that the training loss consistently decreases to near zero, while the validation loss fluctuates significantly and increases steadily after approximately 100 epochs. This indicates that the model memorized the training data but failed to generalize to new, unseen data. On the right plot, the training accuracy reaches 100%, but the validation accuracy stagnates around 50-55%, highlighting the inability of the model to adapt to the validation set.

The poor generalization is likely due to the relatively small dataset size, as transformers typically require large-scale data to perform effectively. Additionally, the complexity introduced by handling three separate streams might have contributed to the unstable validation results.

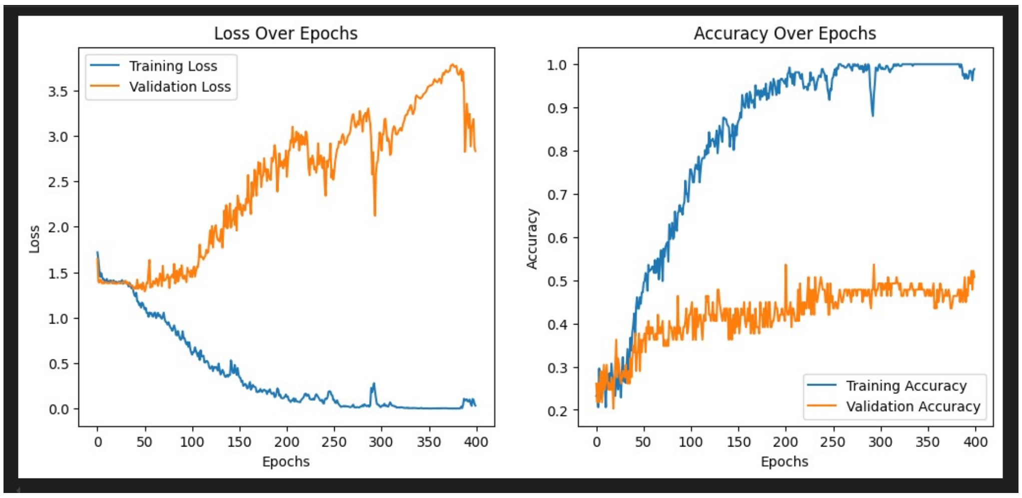


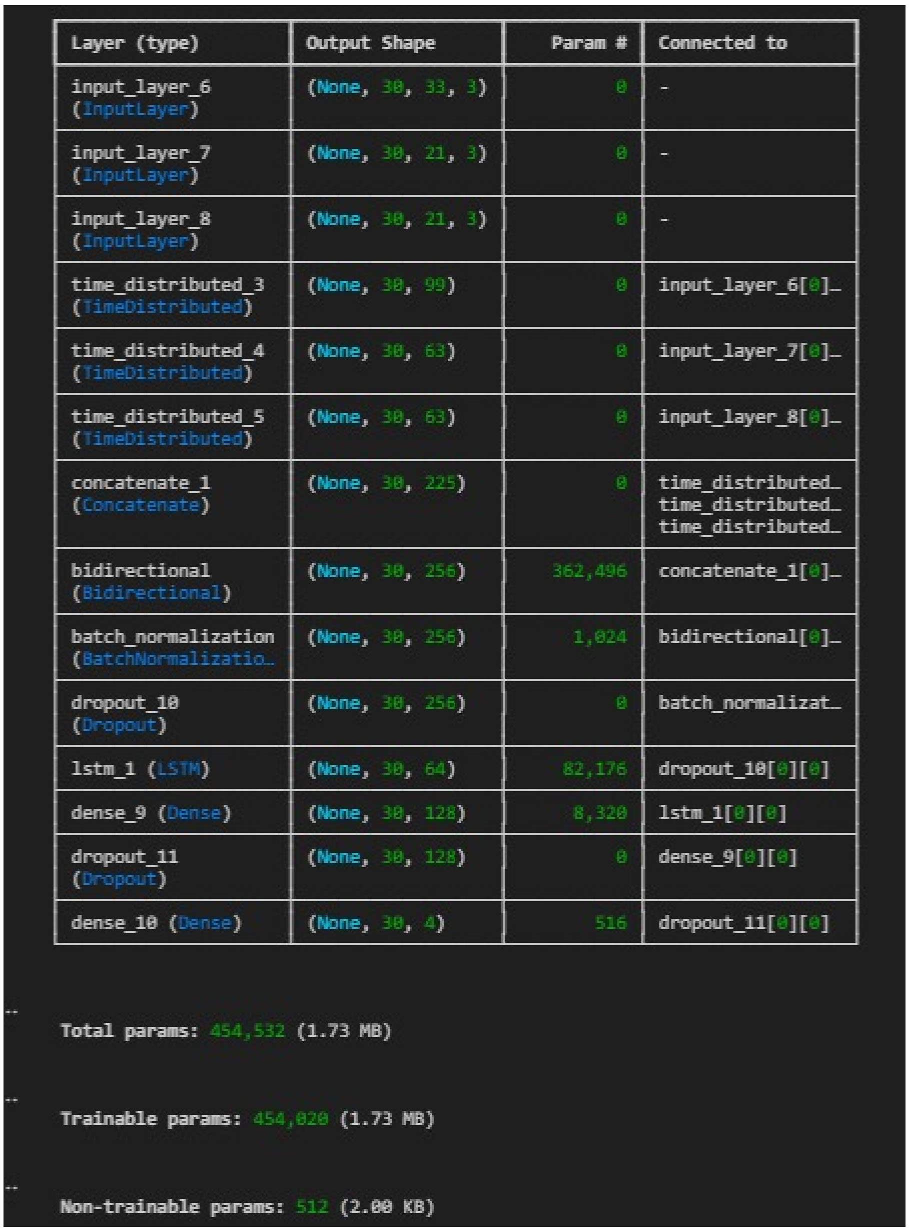
Figure 3.4: Training and Validation Loss/Accuracy for the Multistream Transformer. Severe overfitting is evident as the validation performance deteriorates while training accuracy approaches 100%.

LSTM with Attention

The LSTM with attention architecture employs a bidirectional LSTM to process the frame sequences, followed by an attention mechanism that focuses on the most critical frames. This allows the model to emphasize key temporal patterns, enhancing accuracy by selectively attending to important segments of the input sequence.

In this configuration, the attention mechanism assigns different weights to each frame, amplifying the contribution of significant frames while diminishing less relevant ones. This approach mitigates the limitations of standard LSTM models, which treat all frames with equal importance, potentially overlooking subtle but essential patterns in sign language gestures.

The LSTM with attention model achieved a test accuracy of **85%**, outperforming both the single-stream and multistream transformer models. This performance indicates the effectiveness of attention mechanisms in improving the recognition of sign language at the word level, even with a relatively small dataset.



```
┏━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)        ┃ Output Shape      ┃    Param # ┃ Connected to      ┃
┡━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━┩
│ input_layer_6       │ (None, 30, 33, 3) │          0 │ -                 │
│ (InputLayer)        │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ input_layer_7       │ (None, 30, 21, 3) │          0 │ -                 │
│ (InputLayer)        │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ input_layer_8       │ (None, 30, 21, 3) │          0 │ -                 │
│ (InputLayer)        │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ time_distributed_3  │ (None, 30, 99)    │          0 │ input_layer_6[0]… │
│ (TimeDistributed)   │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ time_distributed_4  │ (None, 30, 63)    │          0 │ input_layer_7[0]… │
│ (TimeDistributed)   │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ time_distributed_5  │ (None, 30, 63)    │          0 │ input_layer_8[0]… │
│ (TimeDistributed)   │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ concatenate_1       │ (None, 30, 225)   │          0 │ time_distributed… │
│ (Concatenate)       │                   │            │ time_distributed… │
│                     │                   │            │ time_distributed… │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ bidirectional       │ (None, 30, 256)   │    362,496 │ concatenate_1[0]… │
│ (Bidirectional)     │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ batch_normalization │ (None, 30, 256)   │      1,024 │ bidirectional[0]… │
│ (BatchNormalizatio… │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dropout_10          │ (None, 30, 256)   │          0 │ batch_normalizat… │
│ (Dropout)           │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ lstm_1 (LSTM)       │ (None, 30, 64)    │     82,176 │ dropout_10[0][0]  │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dense_9 (Dense)     │ (None, 30, 128)   │      8,320 │ lstm_1[0][0]      │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dropout_11          │ (None, 30, 128)   │          0 │ dense_9[0][0]     │
│ (Dropout)           │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dense_10 (Dense)    │ (None, 30, 4)     │        516 │ dropout_11[0][0]  │
└─────────────────────┴───────────────────┴────────────┴───────────────────┘

 Total params: 454,532 (1.73 MB)

 Trainable params: 454,020 (1.73 MB)

 Non-trainable params: 512 (2.00 KB)
```

Figure 3.5: LSTM with Attention Model Architecture.

LSTM without Attention

In this architecture, the model employs a bidirectional LSTM without an attention layer. The frame sequences are processed sequentially, and the final hidden state is used for classification. This simpler approach, although lacking the attention mechanism, surprisingly outperformed the LSTM with attention, achieving a test accuracy of **90%**.

The superior performance of the LSTM without attention can be attributed to several factors:

- **Smaller Dataset:** Attention mechanisms generally require larger datasets to learn effectively. With limited training data, the attention layer may introduce unnecessary complexity, leading to overfitting.

- **Reduced Noise Amplification:** Attention mechanisms may amplify minor variations or noise in the data, especially in smaller datasets. Without attention, the LSTM processes frames uniformly, which may prevent the model from overfitting to unimportant details.

Despite its strong performance, the model could potentially benefit from attention mechanisms if more data were available. This highlights the importance of dataset size in determining the suitability of advanced architectures.

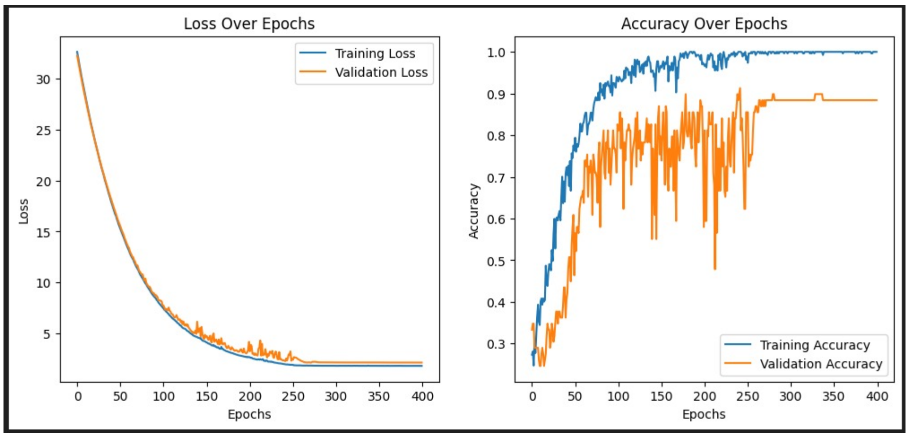
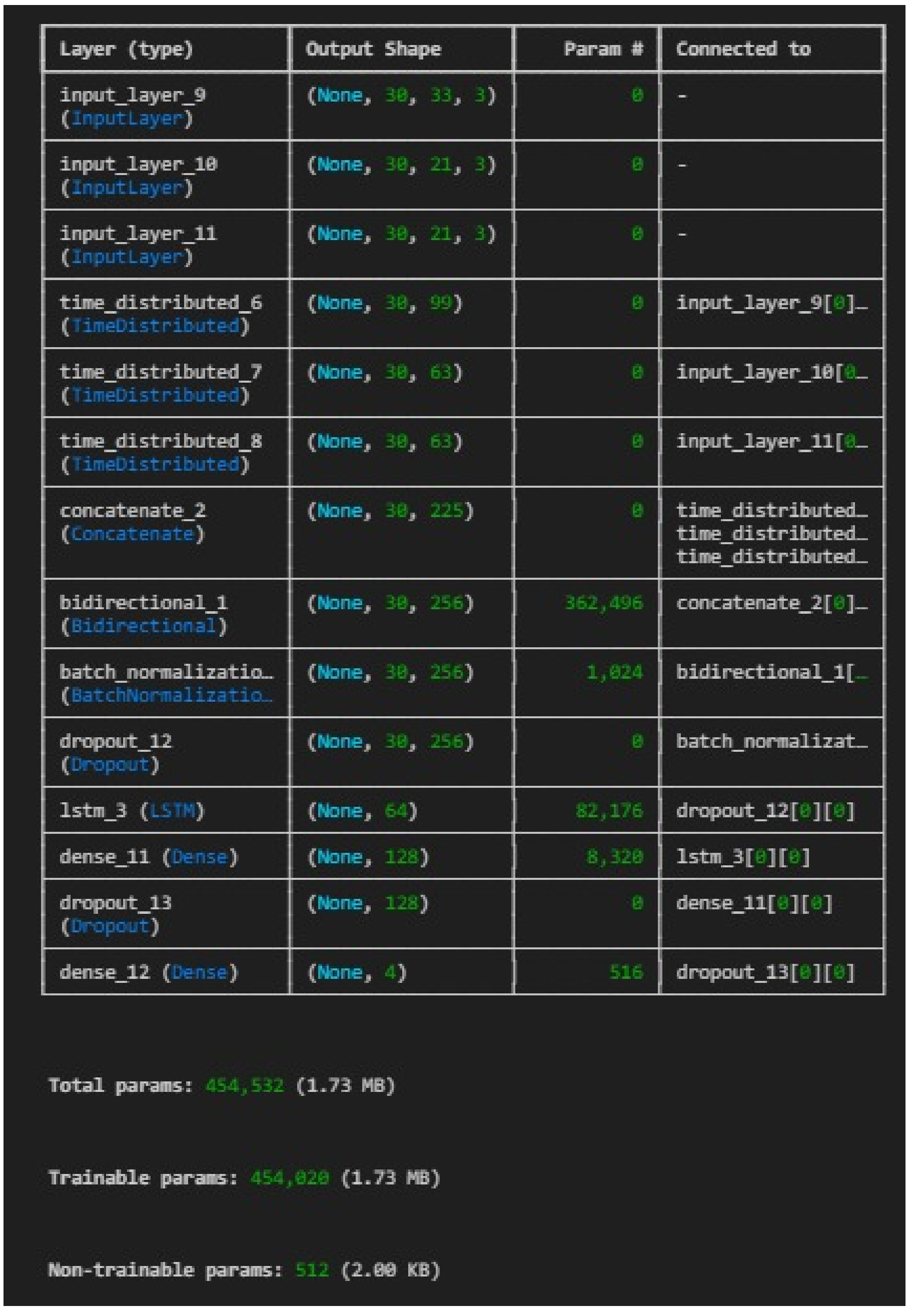


Figure 3.6: LSTM without Attention Model Architecture. Training and validation curves show minimal overfitting, with validation accuracy stabilizing around 90%.



```
┏━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)        ┃ Output Shape      ┃    Param # ┃ Connected to      ┃
┡━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━┩
│ input_layer_9       │ (None, 30, 33, 3) │          0 │ -                 │
│ (InputLayer)        │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ input_layer_10      │ (None, 30, 21, 3) │          0 │ -                 │
│ (InputLayer)        │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ input_layer_11      │ (None, 30, 21, 3) │          0 │ -                 │
│ (InputLayer)        │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ time_distributed_6  │ (None, 30, 99)    │          0 │ input_layer_9[0]… │
│ (TimeDistributed)   │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ time_distributed_7  │ (None, 30, 63)    │          0 │ input_layer_10[0… │
│ (TimeDistributed)   │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ time_distributed_8  │ (None, 30, 63)    │          0 │ input_layer_11[0… │
│ (TimeDistributed)   │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ concatenate_2       │ (None, 30, 225)   │          0 │ time_distributed… │
│ (Concatenate)       │                   │            │ time_distributed… │
│                     │                   │            │ time_distributed… │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ bidirectional_1     │ (None, 30, 256)   │    362,496 │ concatenate_2[0]… │
│ (Bidirectional)     │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ batch_normalizatio… │ (None, 30, 256)   │      1,024 │ bidirectional_1[… │
│ (BatchNormalizatio… │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dropout_12          │ (None, 30, 256)   │          0 │ batch_normalizat… │
│ (Dropout)           │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ lstm_3 (LSTM)       │ (None, 64)        │     82,176 │ dropout_12[0][0]  │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dense_11 (Dense)    │ (None, 128)       │      8,320 │ lstm_3[0][0]      │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dropout_13          │ (None, 128)       │          0 │ dense_11[0][0]    │
│ (Dropout)           │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dense_12 (Dense)    │ (None, 4)         │        516 │ dropout_13[0][0]  │
└─────────────────────┴───────────────────┴────────────┴───────────────────┘

 Total params: 454,532 (1.73 MB)

 Trainable params: 454,020 (1.73 MB)

 Non-trainable params: 512 (2.00 KB)
```

Figure 3.7: LSTM without Attention Architecture.

Chapter 4

Conclusions and Future Work

4.1 Conclusion

This study demonstrates the transformative potential of AI in advancing word-level sign language interpretation. The system primarily focuses on translating video sequences into words by leveraging **Mediapipe** for accurate landmark detection, followed by **LSTMs** to model temporal consistency and an **FNN** for final classification. This pipeline effectively captures the dynamic nature of gestures.

The letter-level implementation served as a foundational stage, enabling effective preprocessing and feature extraction to support the more complex word-level task. By addressing critical challenges such as limited dataset size and hardware constraints, the project establishes a framework for scalable and efficient real-time sign language translation.

While limitations like dataset diversity and real-time deployment speed remain, this integration of state-of-the-art tools exemplifies a significant step toward bridging communication gaps for **Deaf and hard-of-hearing individuals**.

4.2 Limitations

1. **Dataset Quality:** The dataset's limited size and diversity, particularly in the number of videos per word, constrained the model's ability to generalize effectively. This was especially challenging for sequential models like transformers, which require larger datasets.

2. **Hardware Constraints:** The training process was limited by the computational capabilities of the laptops used, affecting both speed and the ability to experiment with more complex architectures or larger datasets.

3. **Real-Time Deployment:** For real-life applications, the system's inference speed must be improved to ensure efficient real-time sign language interpretation.

4.3 Future Work

1. **Dataset Expansion.**

2. **Multimodal Translation:** Incorporate facial expressions and body movements to enhance translation accuracy and context understanding. This multimodal approach can capture the full complexity of sign language.

3. **Language Expansion:** Extend the system to support multiple languages, enabling broader accessibility and impact. Including underrepresented sign languages could significantly increase the system's reach.

4. **System Deployment and Integration:** Focus on deploying and integrating this system into platforms and environments where it can be actively used.

References

A. H. Althubiti and H. Algethami. Dynamic gesture recognition using a transformer and mediapipe. *International Journal of Advanced Computer Science and Applications*, 2024.

T. Ananthanarayana et al. Deep learning methods for sign language translation. *ACM Transactions on Accessible Computing*, 2021.

Mediapipe Documentation. *Real-Time Hand Tracking and Gesture Recognition Framework*. Google Developers, 2023.

A. Hussain et al. An efficient and robust hand gesture recognition system. *Computer Science and Engineering*, 2023.

A. Vaswani et al. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

World Health Organization. Deafness and hearing loss, 2023. URL https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss#:~:text=Overview,will%20have%20disabling%20hearing%20loss. Accessed: Jan. 4, 2025.

M. Zakariah et al. Sign language recognition for arabic alphabets using transfer learning. *Computational Intelligence and Neuroscience*, 2022.