Exploring Sign Language Recognition Through LSTM and Cosine Similarity Modeling

Shanta Maria
BSc in Software Engineering
Department of Computer Science and Engineering
Islamic University of Technology (IUT)
Dhaka, Bangladesh
shantamaria@iut-dhaka.edu

Nafisa Maliyat
BSc in Software Engineering
Department of Computer Science and Engineering
Islamic University of Technology (IUT)
Dhaka, Bangladesh
nafisamaliyat@iut-dhaka.edu

***Abstract*—Integrating deep learning, MediaPipe, and cosine similarity modeling offers a solution to address the challenges in Word-level Sign Language Recognition (WSLR). The research focuses on developing an accurate model for interpreting sign gestures into words, addressing the complexities of fine-grained hand gestures, body movements, and facial expressions. A Long Short-Term Memory (LSTM) model is proposed, utilizing landmarks extracted from sign language videos and undergoing extensive preprocessing. The model architecture incorporates masked input layers, reshape layers, LSTM and dense layers with ReLU activation, batch normalization, and dropout. A cosine similarity loss function is employed for accuracy evaluation. The experiments, conducted on the World Level American Sign Language (WLASL) dataset, reveal challenges in data diversity, overfitting, and computational constraints. The model achieves a testing accuracy of 16.68%, indicating potential for improvement. The paper contributes insights to the field of WSLR, and future work aims to enhance the model's performance, expand its scope to sentence-level recognition, and address real-life applications in the deaf community.**

***Index Terms*—Sign Language Recognition, Deep Learning, MediaPipe, Cosine Similarity Modeling, Long Short-Term Memory (LSTM) model**

GitHub repository: https://github.com/maria-iut1234/ML-Project

I. Introduction

Sign Language is a medium of communication where people can communicate with each other using hand gestures. It is a primary source of communication within the deaf community that helps them to express themselves with those who can speak. However, there is a significant gap between these groups as this requires some sort of translator which may often be inconvenient at times. In response to this, there have been advances in developing various machine learning techniques such as computer vision and deep learning for Sign Language Recognition (SLR). These methods are meant to interpret sign gestures and act as a communication bridge for deaf-mute individuals, empowering them with equal opportunities for personal growth.

According to the World Federation of the Deaf, there are approximately 72 million Deaf people worldwide. More than 80% of them live in developing countries. Collectively, they use more than 300 different sign languages varying across different nations [**1**]. This diversity in sign language across different regions poses a challenge for implementing sign language recognition systems. Furthermore, another major hurdle is the lack of a sufficient dataset. Datasets are usually made by recording new footage or from a small pool of chosen signers, which limits variety [**2**]. Performing automatic SLR is thus more difficult because of the wide vocabulary which poses greater difficulty from the perspective of a machine [**3**].

Recognition of sign gestures at the word level has some challenges. Each word consists of a series of gestures and Word-level Sign Language Recognition (WSLR) requires capturing complex information such as fine-grained hand gestures in a quick motion, body movements, hand shape, facial expression, and the positional relationship among body and both hands [**3**]. There may also be similar gestures that mean different words, which make it difficult to interpret accurately. Even small changes in the signs can mean different words as demonstrated by *Figure 1*. The number of words being used in daily life is huge which further poses difficulty to the scalability of existing recognition systems.

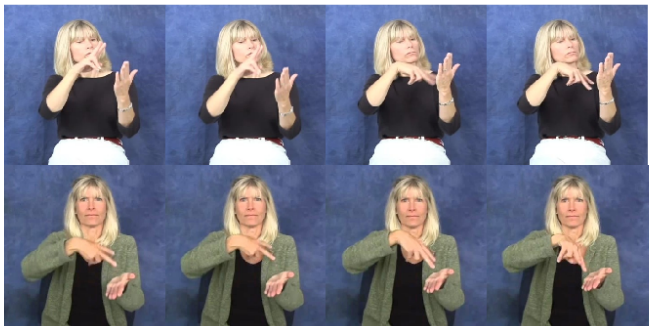


Fig. 1: ASL signs "read" (top row) and "dance" (bottom row) differ only in the orientations of the hands [**4**].

The motivation behind this research is driven by the mentioned complexities and challenges that come with accurately recognizing and interpreting WSLR. This research aims to address the challenges presented by WSLR by exploring the application of deep learning and MediaPipe technology.

Our contribution lies in the use of a model that combines Long Short-Term Memory (LSTM) and dense layers, as well as masking and batch normalization to recognize sign gestures of words through landmark coordinates. Notably, our experimentation uses cosine similarity as a loss function, thus offering a different metric to calculate the accuracy of the model.

II. Background study

A. Sign Language

Deaf and hard-of-hearing (DHH) individuals can use hand gestures and facial expressions to communicate through sign language (SL). These motions, sometimes known as signs, can be represented as sign-words in capital letters and are similar to the building blocks of sign language. We employ an approach known as Isolated Sign Language Recognition (ISLR) to translate individual signs into written glosses to decipher or interpret these signs.

An additional level is known as Continuous Sign Language Recognition (CSLR), which converts a continuous stream of signs into a list of written glosses. It's similar to interpreting the entire exchange using sign language. This can be completed all at once (end to finish) or in phases, by first identifying each sign separately and then assembling them.

Sign Language Translation (SLT) is the process by which we can translate sign language into spoken language to go even further. Either the continuous sign language must first be recognized and then translated into spoken words, or it can be done directly (end-to-end).

Video recordings in color are typically used when recording sign language. However, there are other approaches as well, such as using a skeleton depiction of the signer, finger motions captured by sensors, or depth mapping. Specifically, we use the pose-skeleton technique on ISLR in our work. This means that we create a sort of skeleton map by using models to determine the locations of joints such as the elbows and knees. It facilitates our comprehension of the geographic data in sign language.

To successfully apply this method, we are incorporating MediaPipe into our project. MediaPipe is a flexible architecture that supports multiple modalities. It offers human body identification and tracking pre-trained models, which are essential to our pose-skeleton method. The smooth communication between components is made possible by its graph-based dataflow design and integration with TensorFlow Lite, which guarantees resource efficiency. Our project's overall accuracy and efficiency are improved by MediaPipe's emphasis on efficiency and modularity, which makes it a good fit for pose-skeleton sign language interpretation.

B. Models for ISLR

There are two primary methods for recognizing hand signs: appearance-based and pose-based. Researchers, employing a variety of techniques from traditional to cutting-edge approaches, have sought solutions to the challenges of sign language recognition using these methods. Additionally, both approaches can be tackled by either training a model from the beginning or utilizing pre-existing models specifically designed for sign language recognition to enhance performance. Leveraging pre-trained models proves beneficial in addressing the limited availability of data for sign language analysis.

1) Appearance-Based Approach: In the appearance-based approach, early methods use manual features like Histogram of Oriented Gradients (HOG) or Scale-Invariant Feature Transform (SIFT) for representing hand poses spatially. Classical models like Hidden Markov Models (HMM) handle temporal aspects, and Support Vector Machine (SVM) or kNN classify gestures. Despite initial promise, these models struggle with larger datasets in complex sign language recognition.

Convolutional Neural Networks (CNNs) are notable for training visual data. Recent strategies involve CNNs extracting features from video frames, and then feeding them into a Recurrent Neural Network (RNN) for classification. The approach in using a pre-trained VGG16 model and a stacked GRU for input dependencies.

Another method employs 3D CNNs capturing spatial and temporal features hierarchically. In, 2D filters inflate into 3D for fine-tuning on Kinetics, enhancing temporal and spatial information capture. Features include hand shape, position, and non-manual aspects like eye gaze. The I3D network, ResNet2+1D, and a novel Swin Transformer exhibit notable performance, with the latter leveraging spatiotemporal video correlations.

2) Pose-Based Method: In the pose-based method, recent research utilizes pre-existing pose extractors to gather hand information represented as visual tokens. These tokens embed essential pose details like hand state, temporal features, and chirality. After extracting pose data from tools like OpenHands, subsequent network components may employ attention-based Transformer networks or BERT LSTM networks for sequencing.

Alongside hand poses, some studies introduce skeleton-based techniques in hand sign recognition, effective for isolating dynamic movements. However, acquiring accurate skeleton annotations using motion sensor systems limits available training data. S. Jiang et al propose a novel skeleton graph network capturing dynamic hand and whole-body movements while maintaining spatial features.

Some works focus on extracting body key points and forming a feature vector for RNNs to recognize sign video sequences. Though it captures temporal movements, this method struggles with full spatial information between body key points. Another approach uses a Graph Convolutional Network (GCN) to simultaneously capture temporal and spatial dependencies, representing the human body as a fully connected graph network.

Addressing limited labeled training data, recent strategies include self-supervised pre-training on unlabeled data, leveraging diverse datasets. Models like BERT and SignBERT achieve success through this approach, followed by fine-tuning on labeled datasets. In deep generative models, label omission during training produces accurate predictions when capturing data dynamics and content extensively. Unsupervised techniques, like deep autoencoders, learn to compactly represent spatiotemporal information, using the reconstruction error for classification.

An approach proposed focuses on unsupervised skeleton-based action representation learning. The HiCo framework separates input representation into levels, obtaining features using sequence-to-sequence encoders and downsampling techniques, adaptable for tasks like ASL video recognition with limited labeled data and pre-trained models.

III. Proposed Methodology

Our approach uses an LSTM model with multiple hidden layers for recognition of word-level American Sign Language (ASL). The project followed a process with several steps to ensure the model was unbiased and efficient, as outlined in *Figure 2*.

A. Architecture

Our proposed model architecture is designed to recognize the patterns in sign gestures representing different words of ASL. *Figure 3* shows the layers present in the architecture.

- **Input Layer:** This layer consists of a masking layer to allow variable lengths of inputs which is very important as training, testing and validation data do not have the same number of data counts. The input shape is defined to be (None, 76, 180, 3) which represents 76 frames each containing 180 landmarks with 3D coordinates. It also has a mask value of -80 to ignore values with -80, thus excluding padded values.
- **Reshape Layer:** This layer transforms the data into a three-dimension form of (None, 76, 180*3), which is then passed on to the next layers.

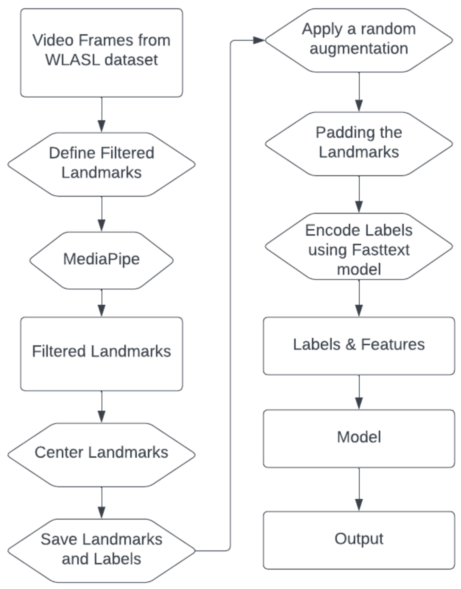

Video Frames from WLASL dataset
Define Filtered Landmarks
MediaPipe
Filtered Landmarks
Center Landmarks
Save Landmarks and Labels
Apply a random augmentation
Padding the Landmarks
Encode Labels using Fasttext model
Labels & Features
Model
Output

Fig. 2: Video frames are initially extracted, and landmarks are filtered. These landmarks undergo centering to mitigate absolute coordinate effects. The processed landmarks are then augmented, padded, and encoded using the FastText model. Serving as input, these encoded landmarks are employed in the model, where the cosine similarity loss function is utilized to determine testing accuracy of the output.

- **LSTM Layers:** Three LSTM layers are used to capture temporal dependencies in data. The layers use ReLU activation functions and are followed by batch normalization and dropout layers for regularization. The first and second layer consists of 64 and 128 units respectively and return sequences. The third layer has 64 units and returns non-sequential data.
- **Dense Layers:** Two dense layers, each of 512 units, are used. These layers use ReLU activation functions as well and similarly, batch normalization and dropout layers are implemented after each dense layer.
- **Output Layer:** The final dense layer of 300 units with linear activation, produces output in the form of (None, 300) dimensional word vectors.

The design is meant to maintain a balance in the level of complexity enough to capture similarity in patterns but not excessively complex that it compromises effectiveness.

B. Dataset

Our research aims to include a model that trains on as many data samples as possible. We have chosen ASL, considering that it is widely adopted by deaf communities in over 20 countries around the world [**4**]. The dataset used for the proposed model is the WLASL-2000 Resized dataset, which is currently the most extensive video dataset for word-level ASL recognition. It contains 21,083 videos that provide

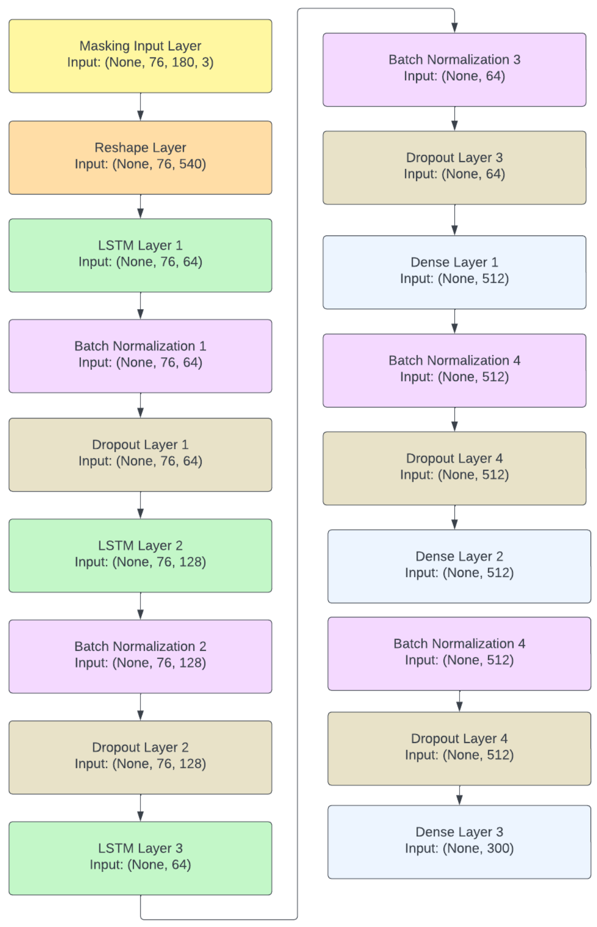


Fig. 3: Architecture of the Proposed LSTM Model

the sign gestures corresponding to 2000 common words in ASL. This substantial amount of data allows for the effective training of the model.

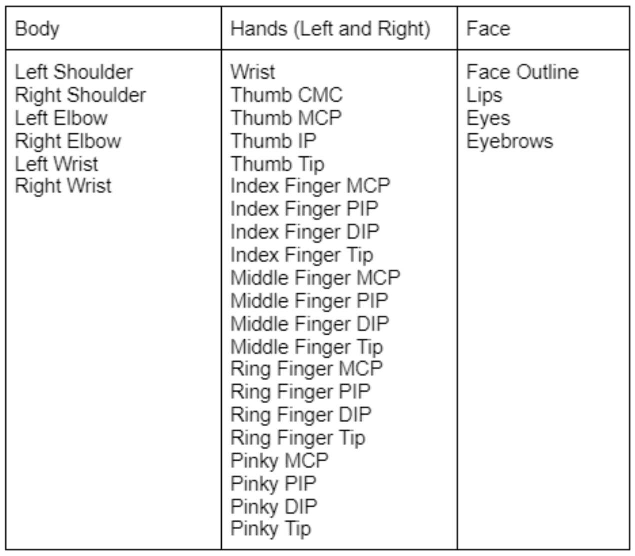
Along with the videos, a WLASL_v0.3.json file is provided, offering all the information required for data processing. It contains a comprehensive gloss in which each entry refers to a unique sign gesture and a list of its video instances. The video instances are also provided in detail such as the start frame and the end frame of the sign gesture as well as how to categorize the data into testing, training or validation data splits. This information was a valuable reference in data processing as it provided reliable labeling for the sign gestures.

C. Feature Engineering

MediaPipe is one of the many machine learning algorithms developed for hand gesture recognition [7]. We used models for tracking hands, pose and face from MediaPipe to obtain landmarks of hands, pose and face of the signers from each frame of the videos, which gives a total of 553 landmarks per frame. The landmark extraction models were applied for processing each frame to produce a sequence of landmarks for each frame.

Due to challenges in memory allocation, we selectively filtered out a subset of these landmarks that contributed to the sign language recognition. Consisting of 180 landmarks per frame in total, the filtered landmarks consisted of all 42 landmarks from the hand as they are crucially important, 6 landmarks from the upper body pose which does not include the face and 132 landmarks from the face. Each landmark consists of a 3D coordinate (x, y, z). This step ensures that only important data is provided to the model thus reducing the noise and shortening the training time of the model.

After the filtering process, a centering step was used to provide uniform and centered features for every frame. The landmarks from hand, pose and face are adjusted separately by taking the first coordinate as a reference point. This step improves the ability of the model to capture the spatial relationship and decreases the variations due to different positions of hands.



Body	Hands (Left and Right)	Face
Left Shoulder Right Shoulder Left Elbow Right Elbow Left Wrist Right Wrist	Wrist Thumb CMC Thumb MCP Thumb IP Thumb Tip Index Finger MCP Index Finger PIP Index Finger DIP Index Finger Tip Middle Finger MCP Middle Finger PIP Middle Finger DIP Middle Finger Tip Ring Finger MCP Ring Finger PIP Ring Finger DIP Ring Finger Tip Pinky MCP Pinky PIP Pinky DIP Pinky Tip	Face Outline Lips Eyes Eyebrows

Fig. 4: Filtered Landmarks

In addition to filtering and centering landmarks, different types of augmentation techniques were used. In particular, rotation around different axes, zooming, shifting, masking, horizontal flipping and downsizing of data by taking every two frames were included. The augmentation methods are shuffled so that one of the mentioned augmentations is applied randomly to the original sample. This introduces variability and more diversity during the training of the model, thus enhancing its ability to adapt and perform better in real-life scenarios.

Lastly, padding is applied to landmarks of a frame which ensures that all samples contain the same number of frames.

A padding value of -80 is applied to samples that have less number of frames than the specified frames (76). The features obtained are then used as inputs in the training of the model.

To apply cosine similarity, an important step involved encoding the labels using FastText encoding to represent the labels in a numerical format. FastText encoding is used due to its reputation for speed and efficiency when handling large amounts of labels. Afterwards, the model was trained using these encoded labels as targets. The combined use of FastText encoding and cosine similarity was used for increasing the interpretability of the sign language recognition model.

D. Model implementation

The implementation of the model uses TensorFlow and Keras to define an LSTM architecture, outlined in *Figure 3*. It contains a masking input layer equipped to handle variable data counts and ignore padded values, followed by a reshape layer. Three consecutive LSTM layers are defined, with varying numbers of units. Additionally, two dense layers are added after the LSTM layer. Each of these five layers uses the ReLU activation function, followed by batch normalization and dropout layers. The final output layer is a dense layer with linear activation.

More compilation configurations of the model were added to further optimize the model. For the loss function, a cosine similarity was chosen to focus on capturing the similarity in the orientation of vectors. For optimization, the Adam optimizer was chosen along with an initial learning rate of 0.05 that exponentially decays at a rate of 0.9 after every 1000 steps. The dynamic learning rate ensures that the training process of the model can adjust accordingly over the epochs. For testing the performance of the model, standard accuracy and cosine similarity metrics were used.

E. Training Procedure

A batch size of 128 was used for training, so the model is trained with 128 samples per epoch during the training process. After each epoch, the weights are adjusted using the calculated gradients and by minimizing the cosine similarity loss function. A validation dataset was used every epoch to monitor the performance of the model. To maximize performance, two callback functions from Keras, ModelCheckpoint and EarlyStopping, were implemented.

During training, ModelCheckpoint was used to save the weights achieved at the end of each epoch in separate .h5 files. Later, the best performance model that was achieved in all the epochs can be retrieved. The callback function only saves the best model weights, determined by the epochs with the highest validation accuracy. This saves space while still providing the option to retrieve the best performance model.

Another callback function EarlyStopping was set to monitor the validation accuracy at the end of each epoch with a patience parameter of 10, which signifies the number of epochs allowed to run without any improvement in validation accuracy before early stopping is triggered. If triggered, the training is stopped before the total number of iterations is reached. The callback function is also configured to restore the best weights which ensures that the model reverts to the weights that produced the highest validation accuracy.

Both these functions contribute to increasing the efficiency of the training process so that only the model weights correspond to the maximum validation accuracy achieved. This approach prevents overfitting, makes the model more generalized towards unseen data and sets up an efficient method to obtain the best model weights.

F. Validation and Evaluation

A validation and test dataset is prepared according to the categorization outlined in the WLASL_v0.3.json provided with the WLASL dataset. The validation dataset was used during the training process and the validation accuracy obtained is used to monitor the model's performance, which prevents overfitting.

The training history is used to store the performance of the model across the epochs and provides insight into the dynamics of training. The model with the best performance obtained is evaluated using a test dataset for calculating the test accuracy of the model. Performance metrics, test loss and accuracy, are used to offer an understanding of how effectively the model would perform on unseen data. The predictions made by the trained model were compared against the true values to determine its accuracy.

The training and validation metrics, loss and accuracy, were visualized using graphs which provided insights into the model's learning process. The graphs displayed the trends of the metrics over the epochs, showing where the model could potentially be improved.

G. Limitations

While our approach demonstrates a possible experimental approach, there were some limitations faced.

It was observed that the model's performance did not have the accuracy level required to consider it suitable for real-life applications. The complications of machine recognition of WSLR might be one of the factors that contributed to this. The model is trained on WLASL only, which may limit its ability to be applied to videos of signs with more diverse settings such as lighting, angle of the signer's position and diverse ways of different signers signing the same word. This may make the model less effective when it comes to data that might contain different variations or unknown gestures that were not in the training dataset.

Although some feature engineering methods were applied, the process can still be improved by refinement of augmentation techniques to introduce more possible variations in sign language gestures.

IV. Experimental Method and Analysis

A. Data Augmentation

A significant challenge in this project is the limited amount of available diversified data and time, which makes it difficult to improve the model's performance. To overcome this challenge, we use data augmentation techniques to enhance the model's ability to recognize different aspects of the data. Recognizing that sign language has inherent variations, we include data from various perspectives to help the model better adapt to new examples.

We utilize diverse data augmentation methods, including random scaling (within a scale factor range of 0.8 to 1.2), squeezing, rotation about the x, y, and z axes, horizontal and vertical shifting, Boolean masking a percentage of the data, horizontal flipping of the 3D skeletal key points we extract, and downsizing the frame count. However, for computational and memory constraints, we randomly select one augmentation technique from the mentioned options. This limitation applies consistently across all dataset subsets to ensure fairness, as performing more than one augmentation technique is not feasible for the entire 2000 dataset.

Introducing data augmentation into our model pipeline helps prevent overfitting and improves the model's generalization.

B. Experiment Design

We conduct experiments using subsets of the WLASL-2000 Resized dataset, specifically with sample sizes of 50, 200, 1000, and 2000. The determination of the subset is governed by the max_labels parameter during the loading of data into the training, testing, and validation sets from their corresponding landmark dictionaries. For each experiment, the subset size is manually specified, and the entire model is rerun to generate distinct model analysis graphs for subsequent comparison.

C. Reporting of Model Accuracy

The resulting accuracies from these optimizations are presented in the *Table I*.

The graphs in *Figure 9* show that as we include more samples in the subset, the overall accuracy decreases. This could be because with a smaller sample size, there's a risk of overfitting, and this effect lessens with a larger sample size,

TABLE I: Test Accuracy For Different Subsets

Subset Sample Size	Test Accuracy (%)
50	22.378
200	21.154
1000	17.964
2000	16.678

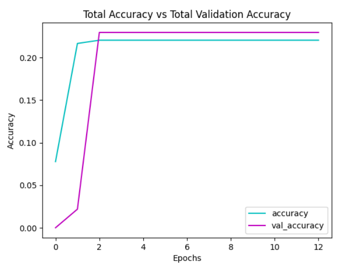


Fig. 5: For 50 samples

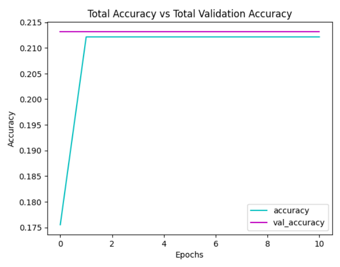


Fig. 6: For 200 samples

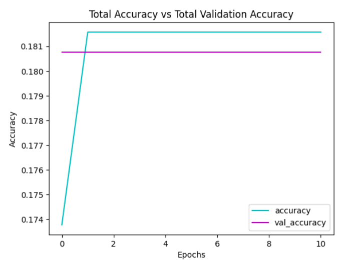


Fig. 7: For 1000 samples

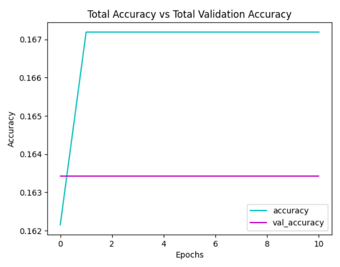


Fig. 8: For 2000 samples

Fig. 9: Total Accuracy vs Total Validation Accuracy

resulting in lower overall accuracy.

D. Reporting of Model Loss

The loss function used for this model is the cosine similarity loss function. Cosine similarity loss is widely used in machine learning, especially in natural language processing. It measures how similar two vectors are by looking at the angle between them, focusing on direction rather than size. The range is -1 to 1, indicating perfect anti-correlation to perfect correlation.

$$similarity(A, B) = cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

When used in a loss function with transformations (like subtracting from 1), negative losses may occur, broadening the range beyond 0. This helps capture a wider spectrum of vector relationships. The goal during model training is to minimize this loss. It encourages the model to learn similar representations for the same class and dissimilar representations for different classes.

The resulting losses from these optimizations are presented in the *Table II*.

The presented graphs indicate an increase in overall loss as the subset size increases. This observation suggests that a smaller sample size may incur a risk of overfitting, leading to

TABLE II: Test Loss For Different Subsets

Subset Sample Size	Test Loss
50	-0.29941
200	-0.31931
1000	-0.36671
2000	-0.35000

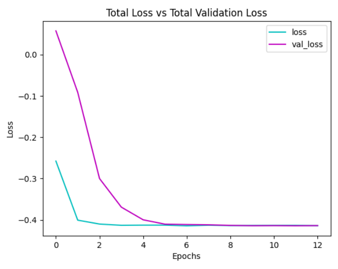


Fig. 10: For 50 samples

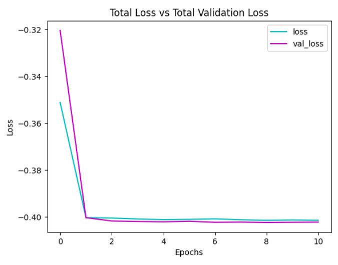


Fig. 11: For 200 samples

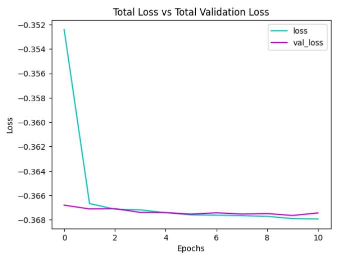


Fig. 12: For 1000 samples

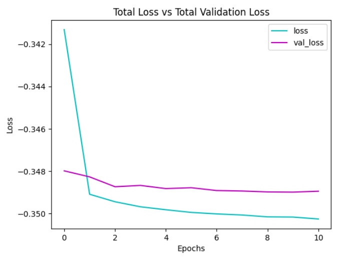


Fig. 13: For 2000 samples

Fig. 14: Total Loss vs Total Validation Loss

lower errors, particularly evident when utilizing the validation set. Conversely, as the sample size increases, the impact of overfitting diminishes, resulting in an increase in overall loss.

E. Effect of Model Parameters

We conducted experiments to assess the impact of various model parameters. Altering the patience parameter from 50 to 10 did not yield significant changes; however, it resulted in the model stopping after 10 consecutive identical total validation accuracy results. This adjustment primarily affected the duration of model execution rather than the overall accuracy, given the inherent fluctuations in accuracy.

Subsequently, we explored the number of epochs, experimenting with values of 1000, 100, and 10. An epoch of 1000 led to premature termination at epoch 54 due to a decline in total validation accuracy from the best-saved weight. Conversely, an epoch of 10 proved inadequate, particularly for a sample size of 50, as the model completed iterations before reaching the best weight. Therefore, a balanced value of 100 epochs was chosen, considering computational limitations and the model's ability to converge.

Modifying the learning rate, within the Adam Optimizer framework that employs an initial rate of 0.05 with exponential decay, did not lead to substantial changes in accuracy and loss outcomes. Adam Optimizer is designed to adaptively adjust the learning rates of individual parameters during training, combining the benefits of both momentum and RMSprop techniques. Even when experimenting with a higher learning rate of 0.001, the impact on model performance remained limited.

Batch size experimentation, with the current size set at 128, involved trials with sizes 10 and 32. While there were no significant alterations in model outcomes, a smaller batch size increased running time significantly. Considering resource constraints and time limitations, a larger batch size was preferred for practical implementation.

F. Confusion Matrix

Confusion matrices were used as evaluation metrics for each sample size.

The examination of confusion matrices in *Figure 19* reveals a bias toward a specific label for each sample size. This tendency is attributed to the cosine similarity between the predicted result and the closest word to the prediction in the dictionary. A more conclusive understanding of the impact of the cosine similarity function, and the potential benefits of utilizing word vectors instead of label encoding as the model output, could be obtained with additional time and computational resources.

G. ROC and AUC

The assessment of the model includes the utilization of Receiver Operating Characteristic (ROC) curves, serving as an evaluation metric. ROC curves, accompanied by their corresponding Area Under the Curve (AUC) values, have been generated for each sample size and unique class label. The resulting ROC curves for different sample sizes are presented in *Figure 24*.

From the figures in *Figure 24*, the observed pattern of AUC values around 0.5 for most labels implies predictions akin to random chance. Notably, the AUC exceeding 0.5 for the '**chair**' label in the sample size of 50 suggests improved prediction accuracy. However, the AUC below 0.5 for the '**man**' label indicates a suboptimal performance in predicting this specific label. It's worth considering that this outcome may be indicative of an overfitted model, contributing to the observed variations in prediction accuracy.

H. Justification of the Results

The final result of the model for each sample size can be observed from *Table I*.

The table indicates a trend where as the sample size grows, the accuracy of the model decreases. This trend might be

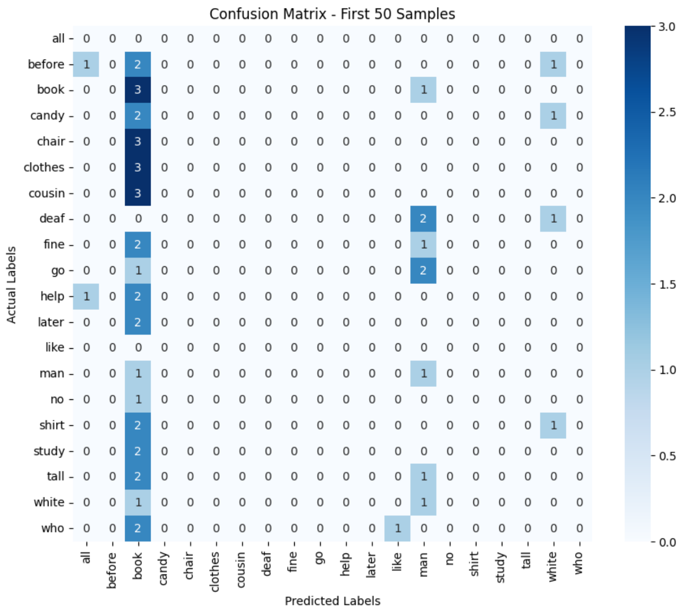


Fig. 15: For 50 samples

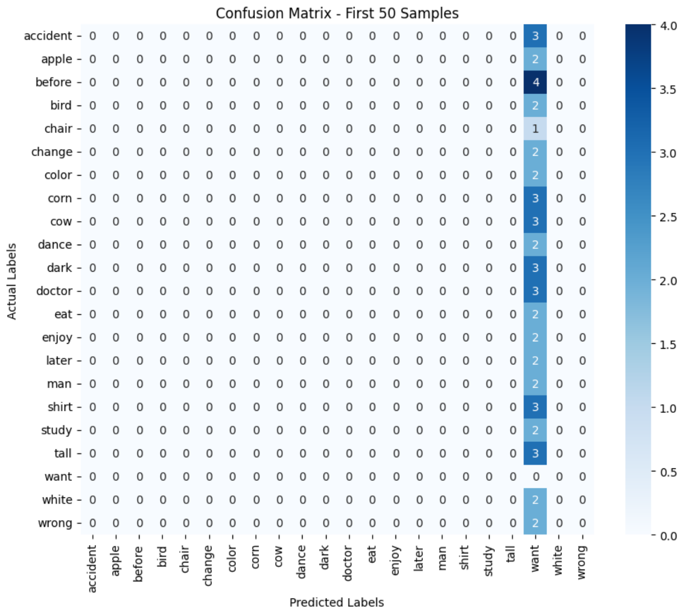


Fig. 16: For 200 samples

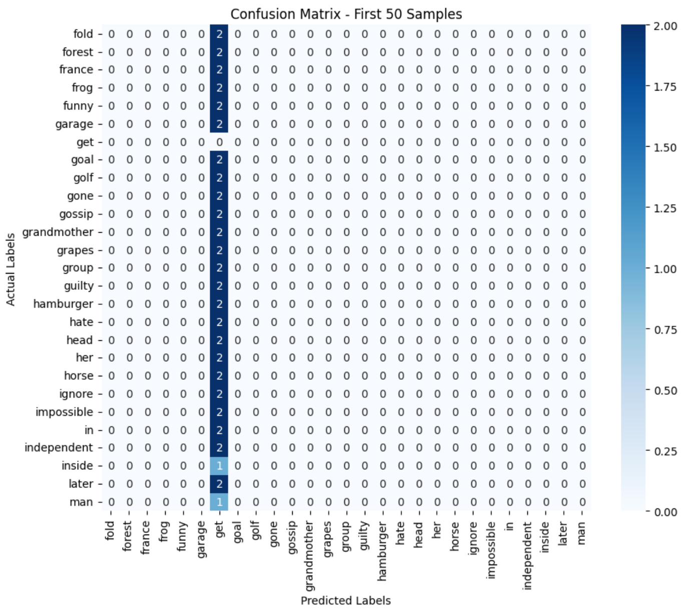


Fig. 17: For 1000 samples

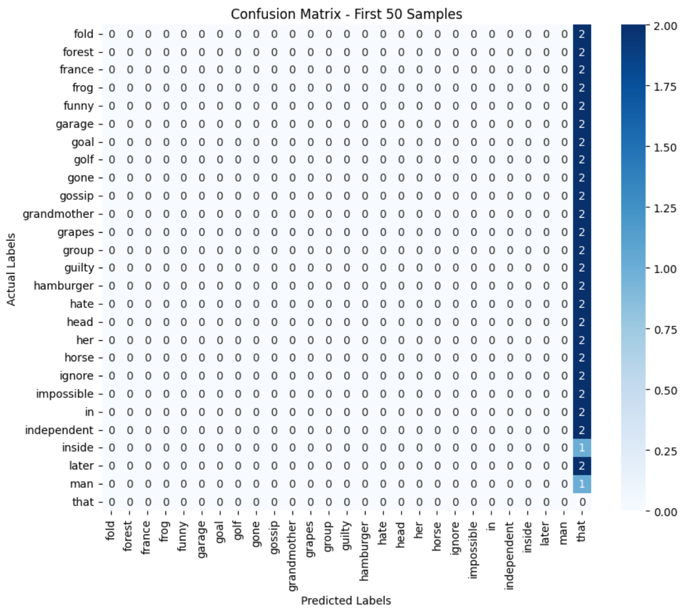


Fig. 18: For 2000 samples

Fig. 19: Confusion Matrices

attributed to the development of a less overfitted model with increasing sample size. However, it's important to note that the model's running time also experiences an increase. Due to limitations in computational resources and time constraints, certain measures such as extensive data augmentation, running the model with an increased number of epochs and patience, and using a less padded version of landmarks as model inputs were not feasible. Considering these constraints, the obtained results exhibiting limited accuracy can be justified and show potential for improvement.

V. Conclusion

In this paper, we presented a fresh outlook on the implementation of MediaPipe, deep learning and a cosine similarity loss function for Word-level Sign Language Recognition (WSLR). While our model has currently achieved a 16.68% testing accuracy on the whole dataset, making it unsuitable for immediate real-life applications at this stage, this project is an important stepping stone

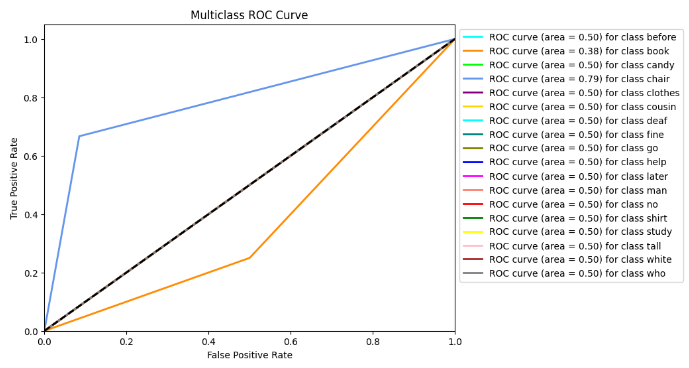


Fig. 20: For 50 samples

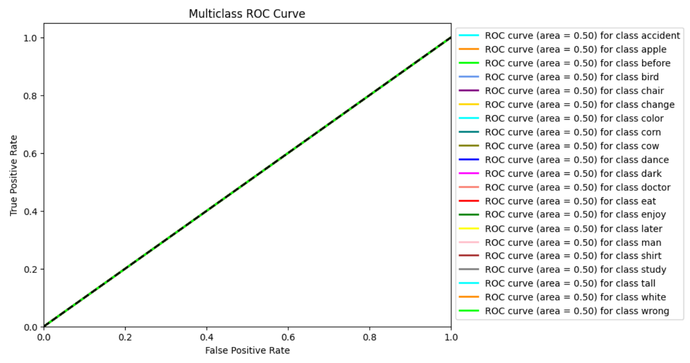


Fig. 21: For 200 samples

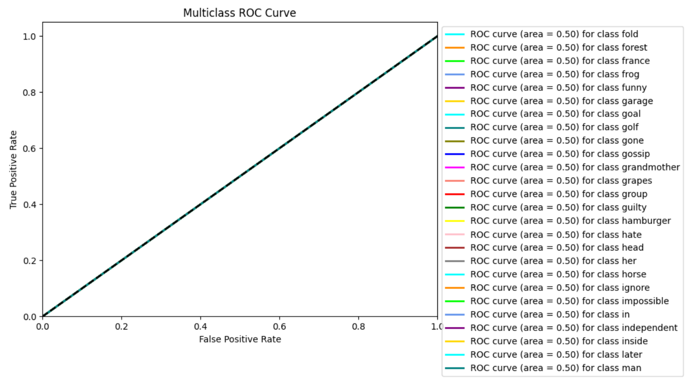


Fig. 22: For 1000 samples

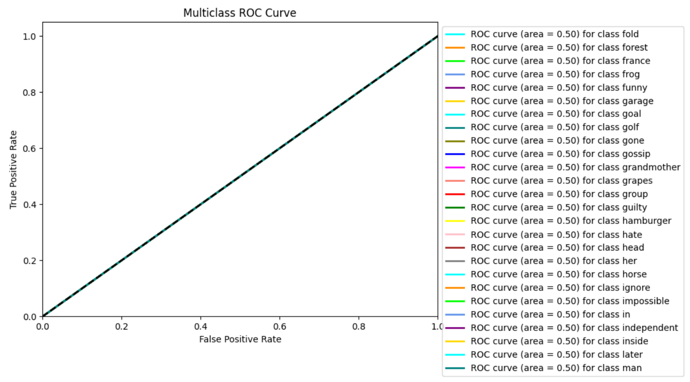


Fig. 23: For 2000 samples

Fig. 24: ROC curves with AUC

to comprehend the challenges posed by word-level Sign Language Recognition.

Some enhancements that could improve our current model such as by implementing advanced data preprocessing and additional augmentation methods. The resource constraints, particularly the availability of computational power, have made this not feasible to implement at this stage. One potential improvement could be the use of a larger dataset that could further boost the model's accuracy by combining data from diverse sources. Additionally, more complex model architectures can be used that can better capture the pattern in the gesture and provide better results.

Our future work will be to overcome these limitations and explore more different methodologies to improve the model's performance. This may include using a mix of more diverse datasets to boost the robustness of the model. Experimenting with more complex architectures along with more advanced techniques such as attention mechanisms [**5**] or transformer-based models [**6**] might produce better results. For our future work, we aim to improve the accuracy of the model and broaden its scope to include sentence-level sign language recognition as well so it can be applied in daily life.

References

[1] "International Day of Sign Languages," United Nations, https://www.un.org/en/observances/sign-languages-day (accessed Dec. 29, 2023).

[2] D. Uthus, G. Tanzer, and M. Georg, "YouTube-ASL: A Large-Scale, Open-Domain American Sign Language-English Parallel Corpus," 2023.

[3] M. Maruyama et al., "Word-level sign language recognition with Multi-Stream neural networks focusing on local regions and skeletal information," SSRN Electronic Journal, 2022. doi:10.2139/ssrn.4263878

[4] D. Li, C. R. Opazo, X. Yu, and H. Li, "Word-level deep sign language recognition from video: A new large-scale dataset and methods comparison," 2020 IEEE Winter Conference on Applications of Computer Vision (WACV), 2020. doi:10.1109/wacv45572.2020.9093512

[5] A. Abdullah, N. A. Abdul-Kadir, and F. K. Harun, "An optimized cosine similarity-based attention gate for temporal sequence patterns recognition," Journal of Physics: Conference Series, vol. 2622, no. 1, p. 012012, 2023. doi:10.1088/1742-6596/2622/1/012012

[6] D. R. Kothadiya, C. M. Bhatt, T. Saba, A. Rehman, and S. A. Bahaj, "Signformer: DeepVision Transformer for sign language recognition," IEEE Access, vol. 11, pp. 4730–4739, 2023. doi:10.1109/access.2022.3231130

[7] L. Chandwani et al., Gesture based sign language recognition system using Mediapipe, 2023. doi:10.21203/rs.3.rs-3106646/v1

[8] P. Selvaraj, G. Nc, P. Kumar, and M. Khapra, "OpenHands: Making sign language recognition accessible with pose-based pretrained models across languages," Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2022. doi:10.18653/v1/2022.acl-long.150

[9] B. Fang, J. Co, and M. Zhang, "DeepASL: Enabling Ubiquitous and Non-Intrusive Word and Sentence-Level Sign Language Translation," Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems, 2017. doi:10.1145/3131672.3131693

[10] Ariesta, Meita Chandra et al. (2018). “Sentence level In- donesian sign language recognition using 3D convolutional neural network and bidirectional recurrent neural network”. In: 2018 Indonesian Association for Pattern Recognition International Conference (INAPR). doi:10.1109/inapr.2018.8627016.

[11] S. Luong, Video sign language recognition using pose extraction and deep learning models, 2023. doi:10.31979/etd.jm4c-myd4

[12] M. C. Ariesta, F. Wiryana, Suharjito, and A. Zahra, “Sentence level Indonesian sign language recognition using 3D convolutional neural network and bidirectional recurrent neural network,” 2018 Indonesian Association for Pattern Recognition International Conference (INAPR), 2018. doi:10.1109/inapr.2018.8627016