Electronic Journal of Business and Management
ICDBSE Special Issue, 2025 pp. 67-80
e-ISSN:2550-1380-ISSN

Application of Image Processing for Sign Language Interpreter

Ng Thung Xeng
Asia Pacific University of Technology and Innovation
ngthungxeng@gmail.com

Hafizul Azizi bin Ismail
Asia Pacific University of Technology and Innovation
hafizul.azizi@apu.edu.my

Yvette Shaan Li-Susiapan
Asia Pacific University of Technology and Innovation
shaan@apu.edu.my

Lau Chee Yong
Asia Pacific University of Technology and Innovation
laucheeyong@apu.edu.my

Sathish Kumar Selva Perumal
Asia Pacific University of Technology and Innovation
sathish@apu.edu.my

Ittidej Moonmangmee
Sukhothai Thammathirat Open University
ittidej@gmail.com

Abstract

The aim of this research is to develop a sign language interpreting system that interprets and translates American Sign Language (ASL) into English words and sentences through machine vision and machine learning. In the proposed methodology, algorithms for data augmentation and data preprocessing as well as model training and evaluation are developed along with the system's Graphical User Interphase (GUI). 3 different models are trained while developing the system, LSTM, Bi-LSTM and GRU and among them, GRU achieves the highest training accuracy of 95.14% and evaluation accuracy of 95.56% hence it is implemented into the system. By testing it in real-time, the system is able to make predictions in 0.143 seconds with 98.79% confidence. Through other various tests, the system proved its capability to produce equally accurate predictions in real-time regardless of the signer's position, distance and hands used. The system is also able to translate ASL sentences into grammatically sound English sentences through OpenAI API.

Keyword: *LSTM, GRU2, American Sign Language 3, Bi-LSTM, Signal Processing*

1.0 Introduction

Sign language is a complex form of communication used by deaf people around the world. There are over 300 different sign languages, each with its own unique signs and grammar. Sign languages are developed from existing spoken languages, but they do not express words directly. Instead, sign languages express meaning through gestures. Sign gestures are divided into two types: static and dynamic. Static gestures involve only handshapes, while dynamic gestures involve both handshapes and body movements. Dynamic gestures are used to represent both words and phrases (Roy et al., 2019). One of the biggest challenges faced by the deaf community is communication barriers. Many hearing people do not know sign language, which can make communication difficult. In the United States, for example, less than 1% of the population knows ASL. There are existing systems that can translate sign language into words (More et al., 2018). These systems use sensors to track hand movements and predict what is being signed. However, these systems can be complex and uncomfortable to use (Anthoniraj et al., 2021). This project aims to develop a sign language translator that is easier to use. The translator will use machine vision and AI to interpret ASL and translate it into English words. This will allow people with limited ASL knowledge to communicate with deaf people more easily (Swamy et al., 2014).

Objectives:

- To design a vision-based sign language algorithm with machine vision and key point detection tools.
- To develop a sign language interpreting model by training it with datasets collected online.
- To evaluate the accuracy of the sign language translator by testing it with real-time data.

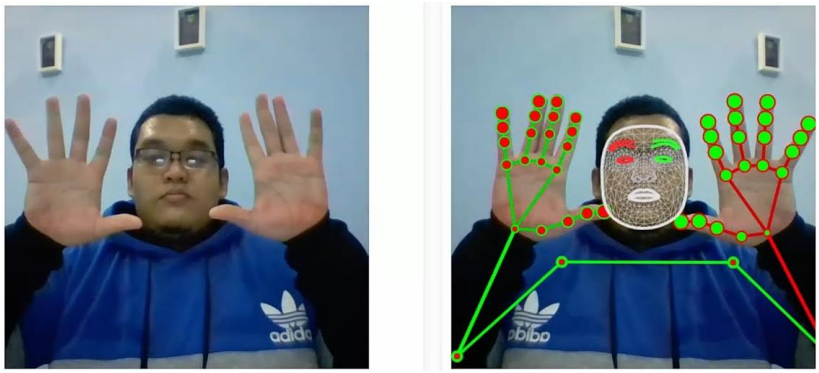
A study from 2019 shows that 25% of deaf adults are diagnosed with depression or anxiety disorder. This is often due to social isolation, as many hearing people do not understand sign language. The sign language translator developed in this project could help to reduce social isolation for deaf people (Vedak et al., 2019).

2.0 Proposed Methodology

2.1 Feature Extraction

To extract features regarding the position, motion, shape and orientation of the signer's hands and arms, an open-source framework developed by Google called MediaPipe will be used. Within the MediaPipe framework, there are pretrained models that could detect the key-points of a person's body, hands, and face in a 3-dimensional space as shown in Figure 1.

Figure 1: **Key-point detection with MediaPipe**



By performing feature extraction with MediaPipe pretrained models, the X, Y and Z coordinates of the key-points will be returned as a numpy array. The number of key-points for the face are much larger compared to the hands and body. If the key-points of the face are extracted, it will increase the size and complexity of each training data. This would in turn increase the time and computational cost for training the model. Hence, in this project, only the key-points of the body and hands of the signer will be extracted. Furthermore, the facial expression of the signer is of less significance compared to the motion and shape of the hands and arms of the signer (BABBAR & LAU, 2020).

There are 21 key-points that can be detected by the MediaPipe hands detection model on each hand. Since the coordinates of the key-points are in 3-dimensional space, there is a total of 126 data points for both hands as shown in Figure 2. For the body, 33 key-points can be detected by the MediaPipe pose detection model. To simplify the training data, only the key-points of the upper body including the arms and shoulders are extracted since the lower body is not important when interpreting sign language. In order to reduce redundant data, the key-points of the hands are not extracted since a higher resolution of hand key-points are captured by the previous MediaPipe hands detection model (Babbar & Yong, 2023). In conclusion, only 17 key-points will be extracted which includes the facial features, shoulders, elbow, and wrist (Dissanayake et al., 2020). According to the documentation, these landmarks are indexed as 0 to 16 by MediaPipe as shown in Figure 3.

Figure 2: Hand key-points detected by MediaPipe

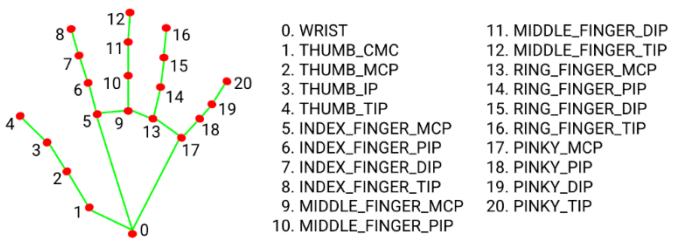
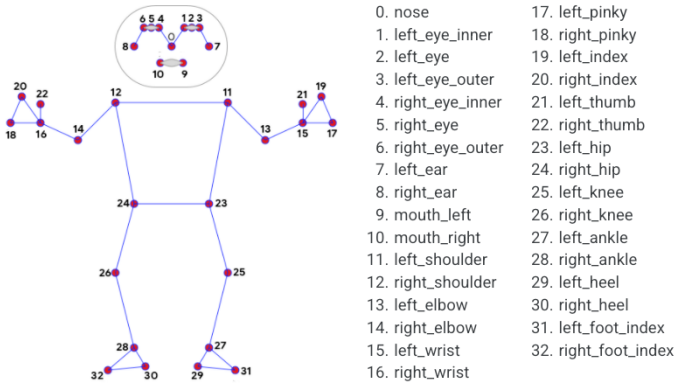


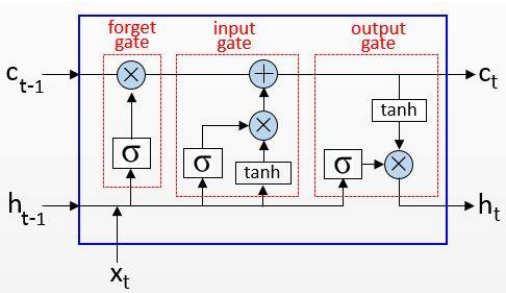
Figure 3: Body key-points detected by MediaPipe



2.2 Concept Design

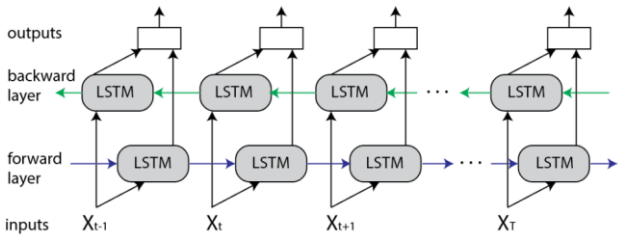
LSTM (as shown in Figure 4) is an extension of RNN with a cell state in addition to the hidden state. The cell state acts as a memory where previous and current information is stored. Information flowing through the cell state is governed by gates, which are composed of sigmoid functions that return values between 0 and 1. There are three gates in an LSTM: forget gate, input gate, and output gate. When input is fed into LSTM, it first passes through the forget gate, which determines whether to keep or discard the previous information based on the new input information. Then, the input gate quantifies the importance of the input information before updating the cell state. Finally, the output gate determines what information will be carried over into the next hidden state, which is used to predict the sequence of data (HUANG & LAU, 2020).

Figure 4: Gates within LSTM



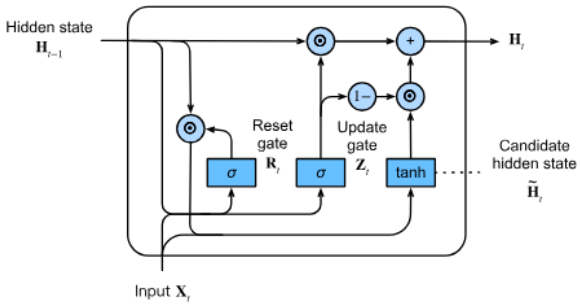
Due to the presence and regulation of cell state, LSTM can make accurate predictions on long and complex sequences. Bidirectional-LSTM (as shown in Figure 5) is made by combining two layers of LSTM. One layer processes the input in a forward direction (past to future), while the other layer processes the input backwards (future to past). This allows the model to preserve both past and future information, which is useful for making predictions on sequential data, especially for natural language processing where word meanings are contextual and depend on both preceding and proceeding words. Bidirectional-LSTM can also be used in sign language translation to produce a more accurate result compared to unidirectional-LSTM. However, due to the extra layer of LSTM in each hidden layer, bidirectional LSTM takes longer to process data compared to unidirectional LSTM (Thea et al., 2022).

Figure 5: Bidirectional-LSTM



GRU (as shown in Figure 6) is an extension of RNN with only two gates. It combines the forget gate and input gate of LSTM into a single update gate, which determines how much current and prior information should be passed to the next hidden state. The reset gate decides how much information to discard. GRU is more compact than LSTM due to its fewer gates, resulting in reduced parameters and processing time while retaining information dependency. However, GRU may not perform as well as LSTM for long-term dependencies or highly complex sequential patterns due to its reduced architecture complexity and parameters (Nath & Arun, 2017).

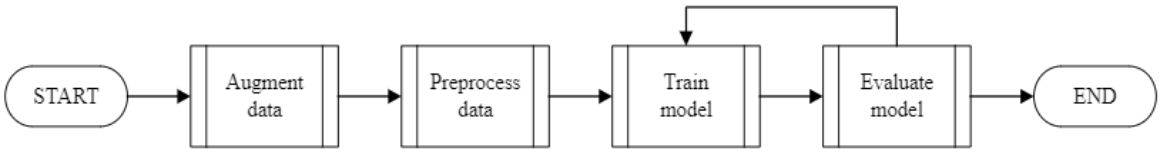
Figure 6: Gates within GRU



2.3 Overall block diagram for model development

Firstly, the dataset is augmented to increase its size and diversity. Then, the augmented data is preprocessed to standardize, clean and organise the data. The preproccesed data is then used to train 3 different models, LSTM, Bi-LSTM and GRU. After that, the models are evaluated with testing data and based on the evaluation and training results, the model's hyperparameters are tuned before it is retrained and reevaluated until a high training and validation accuracy is achieved. After the models are trained and evaluated, the model with the highest validation accuracy will be used to make predictions. The process is Illustrated in Figure 7.

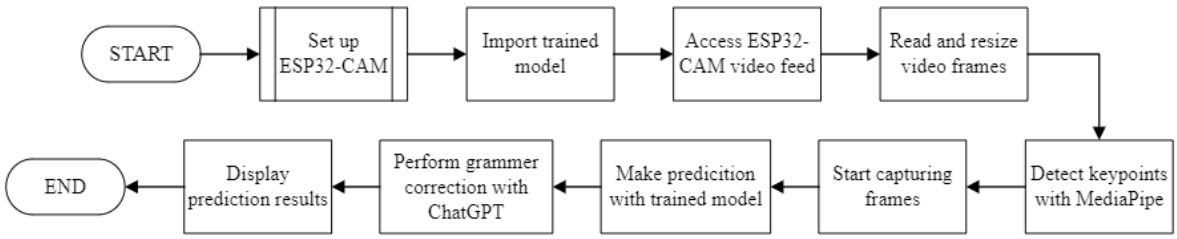
Figure 7: Model Development Block Diagram



2.4 Overall block diagram for implemented sign language interpreting system

Figure 8 is showing the sign language interpreting system block diagram. Before making predictions, the ESP32 camera module is set up in Arduino IDE to capture video frames. Next, the trained model is imported into the system. The video frames captured by ESP32-CAM is then accessed before it is read and resized. Within the resized video frames, the key-points of the signer's hands and body are detected by MediaPipe. On the user's command, the system starts capturing video frames. When sufficient video frames are captured, the trained model makes prediction based on the sequence of video frames. The grammar of the sentence generated is then corrected by ChatGPT before it is displayed to the user.

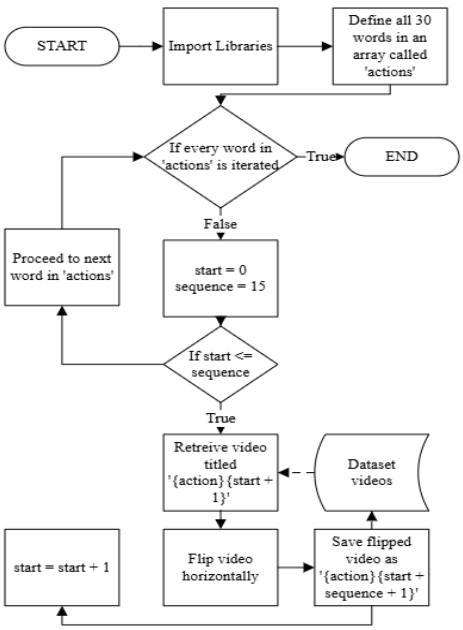
Figure 8: Sign Language Interpreting System Block Diagram



2.5 Data augmentation

Figure 9 shows the logic and process of the data augmentation algorithm. After importing necessary liabraries, all 30 words that the model will be trained to recognise is defined in an array called 'actions'.

Figure 9: Data Augmentation Flowchart



Then, the algorithm enters the first loop in which it checks if every word in the ‘actions’ array is iterated or not. If not, the alrogithm then enters the second loop in which the video corresponding to the current word and index is retrived from the dataset. After retriving the video, it is flipped horizontally and saved back into the dataset as a new video with a new index. Then, ‘start’ is incremented by 1 before the process is looped again. This loop will stop when the value of ‘start’ is more than ‘sequence’ and once that happens, the algorithm proceeds to the next word in the ’actions’ array before returning to the first loop. Once every word in the ‘actions’ array is iterated, the algorithm ends.

Each dataset videos are flipped horizontally and saved into the same dataset. As a result, the size of the original dataset doubles after data augmentation. In total, the dataset contains 900 videos across all 30 words.

2.6 Data preprocessing

The data preprocessing algorithm is shown in Figure 10. It begins by importing libraries and defining the data path. It then enters a loop that iterates through each word in the actions array. For each word, it creates folders named after the word and sequence and generates 30 sequence folders within them. Once all words are iterated, the algorithm proceeds to set up the MediaPipe model for key-point detection.

After setting up, the main loop is initiated, and the algorithm checks if every word in the actions array has been iterated. If not, it enters a nested loop where it retrieves the video corresponding to the current word and sequence from the dataset. It calculates the total number of frames in the video and the frame capturing interval. Then, an inner nested loop is initiated, where the algorithm navigates to the corresponding frame of the video, reads the frame, resizes it, and uses MediaPipe to detect and extract the key-points of the signer within the frame. The extracted keypoints are saved to the data path, and the frame number is updated. This inner nested loop continues until the required number of frames is captured. The outer nested loop then proceeds to the next video, and the entire process repeats until all videos for the current word are processed. When all words in the actions array are iterated, the algorithm ends.

For extract keypoints, the function first places the 3D-coordinates of the detected keypoints into an array called pose. Then, the zero-point and fixed depth are established. The coordinates in pose are made relative to the zero-point and fixed depth, and the results are placed into a new array called pose_zero. After that, the first 17 keypoints in pose_zero are extracted and placed into another array called pose_zero_upper. The same process is repeated for the left and right hand of the signer, where the standardized 3D-coordinates of the detected keypoints are placed into arrays called left_hand and right_hand respectively. Finally, the function returns a concatenated array of pose_zero_upper, left_hand, and right_hand to the main algorithm. The array is illustrated in Figure 11.

Figure 10: Data Preprocessing Flowchart

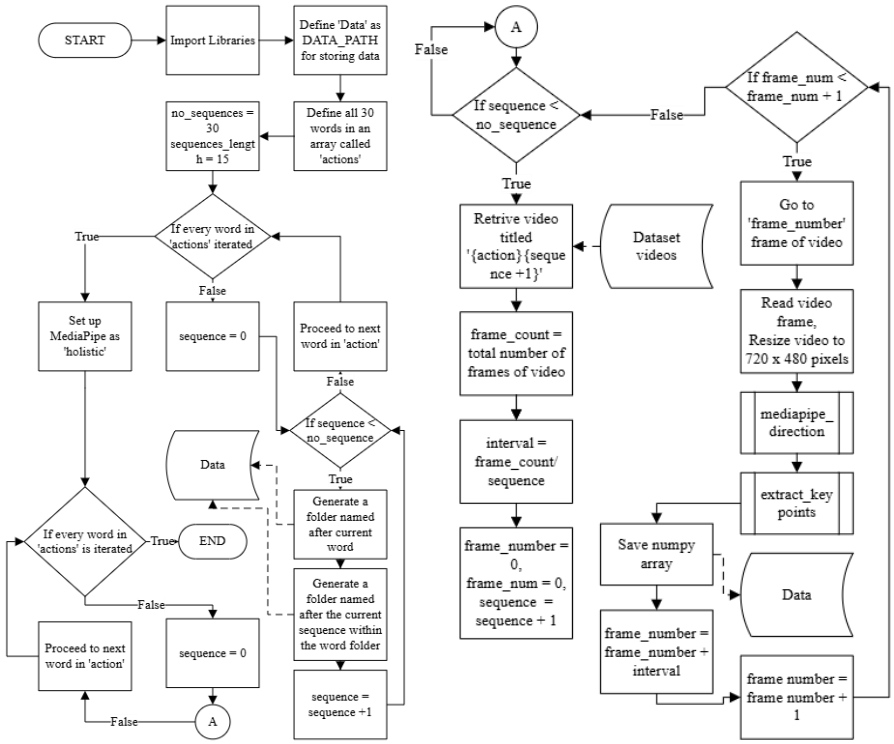


Figure 11: Concatenated Array

```
[ 0.          0.         -0.8         0.01450038 -0.039324   -0.7662444   0.02402705 -0.04018924 -0.76629393  0.03242683 -0.04089651 -0.76627045
 -0.01358199 -0.03783748 -0.76941453 -0.02296695 -0.03733957 -0.76953589 -0.03202605 -0.03644876 -0.76984899  0.04391026 -0.02914982 -0.48626177
 -0.04414633 -0.0232885  -0.50608968  0.01915944  0.03775343 -0.69015698 -0.01725623  0.03718767 -0.69490223  0.13267672  0.21096554 -0.28920554
 -0.11864799  0.20304123 -0.28307109  0.18377036  0.53912127 -0.13768007 -0.19553027  0.50322568 -0.48132085  0.18128085  0.81441897 -0.33463876
 -0.13547641  0.42414564 -1.03052251  0.          0.          0.          0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.          0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.          0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.          0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.          0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.         -0.14876118  0.46611494  0.02227289 -0.15869483  0.49984086  0.00433447
 -0.15749687  0.53449267 -0.00979086 -0.15740401  0.56731683 -0.02131547 -0.16058388  0.58720881 -0.03312427 -0.12014076  0.54029286 -0.01409998
 -0.10475859  0.58457226 -0.02928537 -0.09729528  0.61154217 -0.03870136 -0.09092239  0.63593841 -0.04487386 -0.10030618  0.52632451 -0.012196
 -0.0749746   0.57770443 -0.02531962 -0.06195846  0.60953808 -0.03127067 -0.05205041  0.6358093  -0.03582321 -0.08685079  0.51097775 -0.010713
 -0.06043684  0.55983973 -0.02144561 -0.04822052  0.59180611 -0.02618439 -0.04009971  0.61848378 -0.0295251  -0.07897681  0.49519914 -0.01034685
 -0.05660859  0.53132957 -0.01841587 -0.04728213  0.55623728 -0.02117547 -0.04207429  0.57769668 -0.02318405]
```

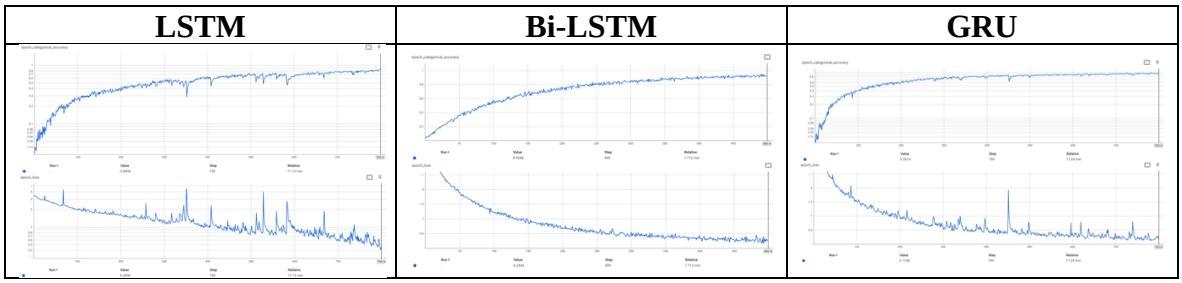
2.7 Model training and evaluation

The algorithm imports libraries, defines the data path, and creates an array of 30 words. It then iterates through each word, labeling it as 'num', and enters a nested loop. Within the inner loop, it loads the corresponding numpy array, appends it to 'sequences', appends the label to 'labels', and updates 'frame_num'. This process continues until 'frame_num' is no longer less than 'sequence_length'. The outer loop then repeats until 'sequence' is no longer less than 'no_sequence'. After the main loop, 'sequences' and 'labels' are stored as 'X' and 'y', respectively. These are then split into training and testing data. The training data is used to train the model, and the testing data is used to evaluate its performance. Finally, the evaluation results are displayed.

By referring to the training results of each model as shown in Figure 12, GRU achieved the highest accuracy of 95.14%. After evaluating each model and analyzing the evaluation matrices, GRU also achieved the highest validation accuracy. Since GRU achieved the

highest training and validation accuracy amongst the 3 model, this model is selected to be implemented in the final sign language interpreting system to make predictions in real-time.

Figure 12: Training Result of Each Model



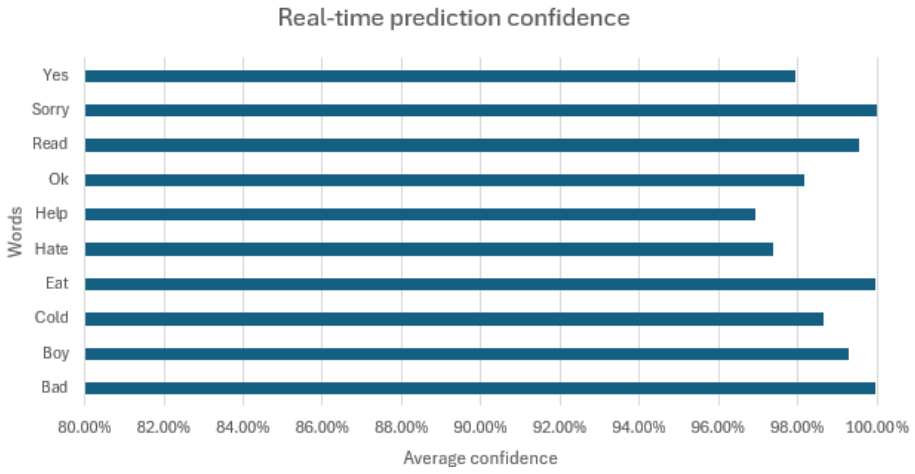
3.0 Testing

In this section, the sign language interpreting system is first be tested on how well it performs in real-time. Next, the robustness of the system is tested by subjecting it under various situations including the varying positions and distance of the signer relative to the camera, the hands used by the signer to perform ASL and the brightness level of surroundings.

3.1 Real-time predictions

For this test, the signer is required to be 1 meter (m) away from the camera and be positioned at the centre of the frame under a brightness level of 40 Lux (lumen per square meter). The signer then signs 10 random words from Table 3.4 and the prediction results are recorded. This process is repeated 2 more times to obtain the average accuracy and confidence of the system across the 10 words. The model managed to predict all 10 words correctly. Based on Figure 5.1, the prediction confidence of the system across all 10 words is at least 96% on average. Overall, the system has a prediction confidence of 98.79% with an average processing time of 0.143 seconds as shown in Figure 13.

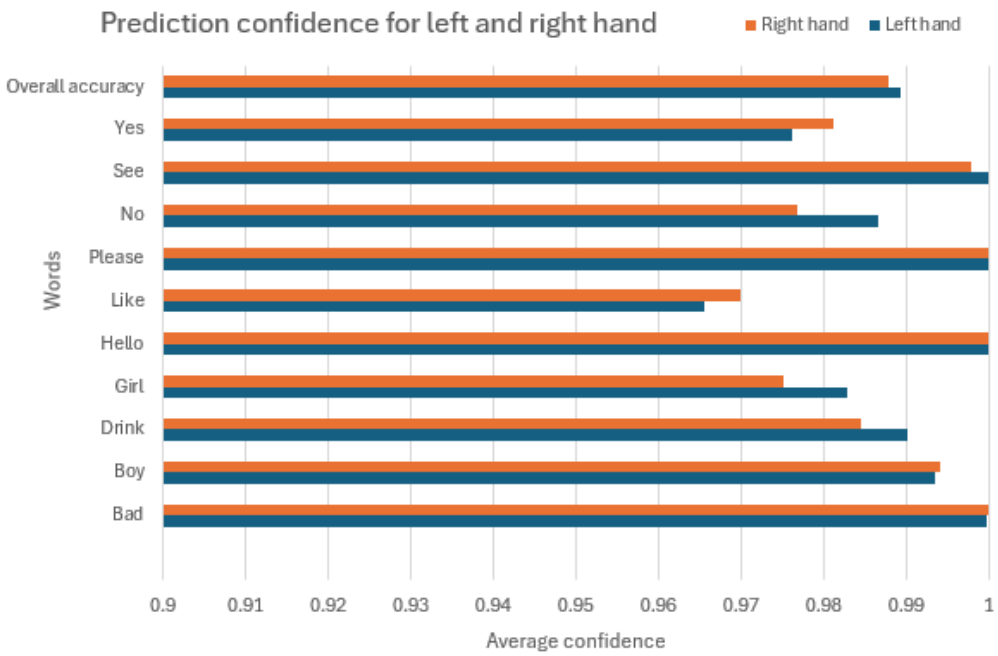
Figure 13: Real-time Prediction Confidence



3.2 Hands Used by Signer

The signer must be 1m away from the camera and positioned at the centre of the frame under 40 Lux. Next, 10 random words that requires only one hand to sign are selected from Table 3.4. The words are then signed by the signer first with his left hand, then again with his right hand. This process is repeated 2 more times to determine the system's average accuracy and confidence across the 10 words as shown in Figure 14.

Figure 14: Prediction confidence for left and right-hand



The model is able to accurately predict all 10 words for left and right hand. Based the results plotted in Figure 5.2, it can be observed that the prediction confidence across all 10 words signed by left or right hand are similar. Amongst the 10 words, 'no' has the biggest difference in terms of confidence, which is only 0.98%. The overall confidence of the system in interpreting words signed by left or right hand only have a difference of 0.15%. In conclusion, this system is capable of interpreting ASL regardless of which hand is being used by the signer.

3.3 Position of Signer

The signer is required to be 1m away from the camera for this test and under 40 Lux. Then, the signer is positioned in the center of the frame and 5 random words from Table 3.4 are signed. Next, the 5 random words are signed again but the signer is positioned on the far left of the frame. This process is repeated once more with the signer positioned on the far right of the frame. The prediction results of the system and its accuracy and confidence are recorded and tabulated throughout this test as shown in Figure 15.

The model proved to be able to make accurate predictions across all 5 words when the signer is in all 3 positions. By referring to the results plotted in Figure 5.3, the prediction confidence across all 5 words when the signer is in varying position are relatively similar, the biggest

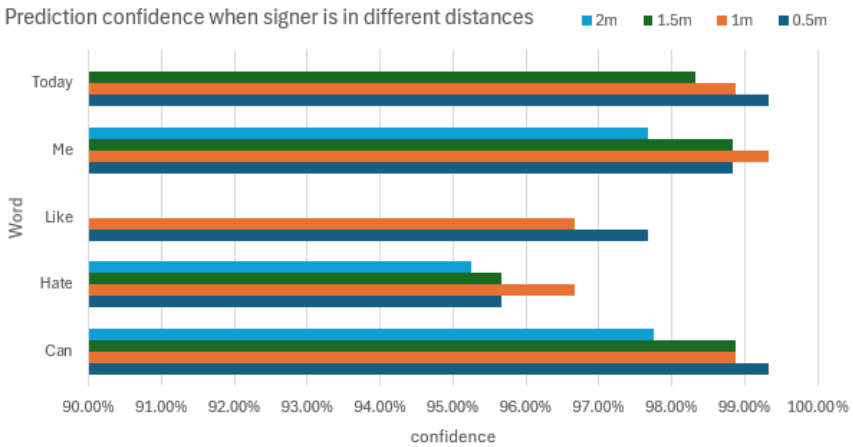
difference being only 0.56%. After this test, it can be concluded that this system is able to interpret ASL regardless of the signer's position within the frame.

Figure 15: Prediction Confidence for Different Positions of Signer

3.4 Distance of Signer

For this test, the signer is required to be positioned at the center of the frame and under 40 Lux. Then, the signer is positioned 0.5m away from the camera before 5 random words from Table 3.4 are signed. This process is repeated with the distance of the signer from the camera changed to 1m, 1.5m and 2m. Throughout this test, the system's prediction results are recorded and tabulated as shown in Figure 16.

Figure 16: Prediction Confidence Different Distance of Signer



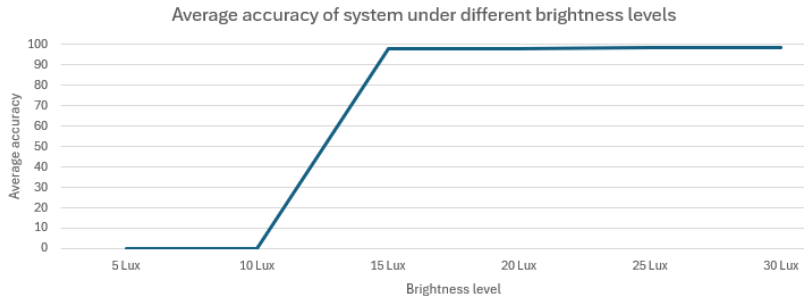
The system is able to produce accurate predictions with over 95% confidence for the words 'me', 'hate' and 'can'. For the word 'like', the system predicts 'please' once the signer is at 1.5m or further whereas for the word 'today', the system predicts 'can' when the signer is at 2m away from the camera. These incorrect predictions are due to the low resolution of the

ESP32 camera module which gives MediaPipe difficulties in detecting the signer's key-points accurately.

3.5 Brightness Level

The signer is required to be 1m away from the camera and be positioned at the center of the frame for this test. Then, 3 random words are signed by the signer under a brightness level of 5 Lux and the prediction results are recorded. This process is repeated with the brightness level incrementing by 5 Lux each time upwards to 30 Lux.

Figure 17: Prediction Confidence Under Different Brightness Level



The system has an average confidence of 0% but at 15 Lux and above, the system's average confidence is about 98%. At low brightness levels, the ESP32 camera module is unable to clearly capture the signer, hence, no key-points are detected by MediaPipe which means no prediction can be made by the system. In conclusion, this system can only perform well under brightness levels of 15 Lux or higher as shown in Figure 17.

Table 1: Accuracy Of Other Systems

Arthur	Model	Accuracy (%)
Stylianos K. Psara	GRU	89.03
Gerges H.Samaan et al.	GRU	100
Shamita S H & Dr. Badarinath K	LSTM	97
Ridwang	GRU	85
Bushra A. Al-Mohimeed et al.	CNN-LSTM	70
This project	GRU	95.56

The accuracy of other models developed by other researchers that was reviewed. The performance of the model surpasses most of researcher's model apart from Gerges H.Samaan et al. and Shamita S H & Dr. Badarinath K in which their model is able to achieve 100% and 97% accuracy respectively. This is because their model is only trained to recognize 10 and 24 words respectively which is significantly less than the 30 words that our model is trained to recognize. The comparison is tabulated in Table 1.

By looking into the systems developed by the researchers, there is a of lack features such as successive predictions and grammar correction which restricts the user from constructing grammatically sound sentences with ASL. Our system however is able to make predictions successively and apply grammar correction to the constructed sentences which makes understanding ASL sentences a lot easier.

4.0 Conclusion and Recommendations

In order to achieve the objectives of this project, a vision-based sign language interpreting algorithm that utilizes OpenCV for machine vision and MediaPipe for keypoint detection is designed. Next, 3 different models, LSTM, Bi-LSTM and GRU are developed and trained with the dataset collected online which yields a training accuracy of 84.58%, 92.08% and 95.14% respectively. The accuracy of the 3 models is then evaluated with testing data and amongst them, GRU achieves the highest accuracy of 95.56%. The GRU model is also tested in real-time under various conditions. Through the conducted tests, it is determined that the GRU model can make predictions in real-time with 98.79% confidence regardless of the signer's position the frame or the hands used. The test results also suggest that for optimal performance, the signer should be 1 meter away from the camera with a minimum brightness level of 15 Lux.

5.0 References

Anthoniraj, S., Ganashree, V., Umdor, B. J. R., Sai, G. D., & Navya, B. (2021). Sign Language Interpreter Using Machine Learning. 2021 International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICAECA),

BABBAR, S. M., & LAU, C.-Y. (2020). Medium term wind speed forecasting using combination of linear and nonlinear models. *Solid State Technology*, *63*(1s), 874-882.

Babbar, S. M., & Yong, L. C. (2023). Short term solar power forecasting using deep neural networks. Future of Information and Communication Conference,

Dissanayake, I., Wickramanayake, P., Mudunkotuwa, M., & Fernando, P. (2020). Utalk: Sri Lankan sign language converter mobile app using image processing and machine learning. 2020 2nd International Conference on Advancements in Computing (ICAC),

HUANG, Z.-J., & LAU, C.-Y. (2020). Camera Vision Foreground Detection Employing Frame Differencing And Background Subtraction In Solar Farm Monitoring Application. *Solid State Technology*, *63*(1s), 883-892.

More, V., Sangamnerkar, S., Thakare, V., Mane, D., & Dolas, R. (2018). Sign language recognition using image processing. *SIGN*, *21*, 22nd.

Nath, G. G., & Arun, C. (2017). Real time sign language interpreter. 2017 IEEE International Conference on Electrical, Instrumentation and Communication Engineering (ICEICE),

Roy, P., Uddin, S. M., Rahman, M. A., Rahman, M. M., Alam, M. S., & Mahin, M. S. R. (2019). Bangla sign language conversation interpreter using image processing. 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT),

Swamy, S., Chethan, M., & Gatwadi, M. (2014). Indian sign language interpreter with android implementation. *International Journal of Computer Applications*, *97*(13).

Thea, S., Lau, C. Y., & Lai, N. (2022). Automated IoT Based Air Quality Management System Employing Sensors and Machine Learning. *Journal of Engineering Science and Technology*, *2022*, 172-184.

Vedak, O., Zavre, P., Todkar, A., & Patil, M. (2019). Sign language interpreter using image processing and machine learning. *International Research Journal of Engineering and Technology (IRJET)*, *6*, 1907-1909.