ASL Detection Using CNN and SAM2

Seoyun Yang
Division of Engineering Science
University of Toronto
Toronto, ON
seoyun.yang@mail.utoronto.ca

Hyunji Kim
Division of Engineering Science
University of Toronto
Toronto, ON
hji.kim@mail.utoronto.ca

Mia Chang
Division of Engineering Science
University of Toronto
Toronto, ON
mia.chang@mail.utoronto.ca

Abstract

This paper tackles the task of real-time American Sign Language (ASL) translation to facilitate communication for deaf individuals. Our approach involves taking multiple snapshots from videos and classifying these images into their corresponding ASL signs using a combination of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). The current classification accuracy achieved is 19.23% on a 10-class dataset, showing significant room for improvement. To investigate the cause of this limited performance, we explored two directions: increasing the amount of training data and implementing a baseline CNN model that uses only a single frame. These approaches provided insights into the challenges of our current model and justified the need for improved preprocessing techniques. In particular, we propose using Segment Anything Model 2 (SAM2) in combination with MediaPipe to enhance spatial clarity and reduce background noise before feeding the frames into the CNN-RNN pipeline.

Code: https://github.com/mmiachang/ece324

1 Introduction

This project aims to address the challenge of live ASL translation by fine-tuning computer vision technologies such as SAM2 and CNN. The goal is to convert ASL into subtitles, enhancing accessibility for ASL users. Currently, there is a significant amount of research being conducted on still images, while challenges remain in effectively detecting and addressing video object segmentation (VOS). The ASL translation problem is primarily divided into two key tasks: segmenting the hand from the video, which involves VOS, and identifying the specific sign language gesture, which involves image classification.

2 Related Work

SAM2 Segment Anything (1), the state-of-the-art image segmentation model, released by Meta, enables interactive segmentation, allowing users to add prompts to easily segment objects and produce the masklet across the entire video. SAM2's core architecture includes a transformer-based image encoder/decoder, a prompt encoder, a mask decoder, and a memory mechanism that enables tracking of objects even when they move out of the frame.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

MediaPipe MediaPipe ([2]) is an open-source framework released by Google Research, providing libraries to quickly apply AI and ML techniques on Android, Web, Python, and iOS platforms. MediaPipe Solutions include LLM inference, object detection, image classification, hand landmark detection, gesture recognition, and more. For the purpose of this paper, the hand landmark detection solution will be used.

3 Methodology

3.1 Data Preprocessing

3.1.1 Dataset

The MS-ASL dataset ([3]) consists of a total of 1,000 classes, with 16,054 training samples, 5,287 validation samples, and 4,172 testing samples. Each "class" corresponds to a word in American Sign Language represented by the videos. The training, validation, and testing datasets is represented as a list of dictionaries, containing the data in the form of :

{original text from the source, normalized text (class), start time of SL, id of the signer, start frame index, end frame index, file name, class number, height of the video, frame per second, end time of SL, url to the video, class, bounding box, width of the video}

The data was not organized by class, necessitating grouping of the samples accordingly when creating batches.

3.1.2 Choice of Batch size and number

For this stage of training, the team decided to use a single batch of 10 classes. The goal is to estimate the training time and assess its cost in terms of time, storage, and GPU load, using the largest batch size that the team's available resources allow. Using 10 classes allows the model to make predictions by providing a sufficient number of possible labels, while ensuring that the training can be completed within a reasonable time frame. This approach takes into account that the data is in the form of videos, which must be converted into images with timestamp, necessitating flexibility to accommodate the possibility of the data size increasing as the team progresses to later stages.

3.1.3 Batching

As the first step in batching, the classes are parsed according to the desired batch size and number of batches. Since there is no specific significance or predefined ordering of the classes, the parsing begins from the start of the list. To create the batches from each dataset, the corresponding data for each class is extracted and saved as a JSON file, which contains a nested structure of dictionaries and lists in the following format:

{batch index, class names, class indices, data}

Using the batch JSON file, the video URLs from the data are accessed, downloaded, cropped, and clipped based on the provided start and end times, as well as the bounding box information. To minimize the memory usage, videos are temporarily saved as a temp.mp4 and deleted after their processed clips are stored in the corresponding class folder. This organization ensures that each video clip is clearly labeled and easily accessible.

3.1.4 Data Splitting

The team encountered an issue where approximately 47.03% of the videos could not be processed due to factors such as URLs no longer being publicly available. This significantly reduced the size of the dataset, particularly affecting the testset, which was already small in the original MS-ASL dataset. Additionally, this led to inconsistencies in the dataset size across classes, potentially introducing imbalance during model training. To address this issue, the team decided to use only the training dataset, splitting it into new training and test sets. To ensure fair distribution of videos, the videos are randomly shuffled within each class before splitting. Due to the limited amount of data, the team didn't not perform a validation, and an 80:20 split ratio was applied between training and test sets.

After the split, the team had an average of 19.6 training videos and 5.4 test videos per class, with standard deviations of 1.65 and 0.3, respectively.

3.1.5 From Video to Tensors

The splitted dataset consists of ten distinct classes, with approximately 30 videos in the training set and 10 videos in the testing set. Each video was segmented at a 30-millisecond interval and cropped to a spatial resolution of 224×400 pixels. Upon analysis, it was observed that most videos start to repeat after 15 frames. Therefore, a fixed sequence length of 15 frames was adopted: videos exceeding this length were truncated and those shorter were padded with black frames. Furthermore, to reduce the dimensionality and increase computation efficiency, all frames were converted to grayscale tensors, as sign language recognition is invariant to colour.

3.2 CNN and RNN

3.2.1 Model Architecture and Training

The model utilizes a CNN encoder to extract spatial features from each frame and convert them into a flattened tensor. Since video is time-series data, an LSTM was employed following the CNN to capture temporal variation across frames. Figure 1 provides a detailed illustration of the model architecture. Due to the complexity of the structure, particularly the CNN encoder—which has approximately 420k parameters to optimize in the best-performing model—mini-batch training was employed to efficiently manage computation and leverage GPU parallel processing.

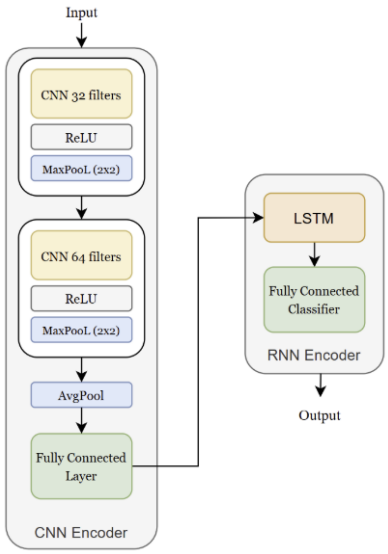

Figure 1: Model architecture.

3.2.2 Results, Challenges and Possible Solutions

Coordinate descent was used for hyperparameter tuning and the results can be found in Table 1. During the training process, a steady increase in training accuracy was observed initially, indicating that the model learned efficiently at the start. However, this rise of training accuracy hit a plateau at around 37%, and the highest testing accuracy achieved was 19.23%. To investigate the cause of our poor performance, we explored two directions. The first was identifying that parts of the training and testing datasets were missing. After recovering the missing data, we re-ran the CNN-RNN model, which slightly improved the best test accuracy to 21.88%. This suggests that our model's performance is significantly influenced by the amount of data available for training. Therefore, one clear direction for improvement is to augment the dataset using techniques such as rotation, translation, or other

Table 1: Coordinate descent search result. Note that the results are presented in the following format: (parameter choice, testing accuracy). The highest testing accuracy achieved was highlighted.

Hyperparameter	Training Accuracy		
Batch Size	(9, 3.85%)	(18, 8.33%)	(27, 13.68%)
Learning Rate	(0.01, 13.68%)	(0.05, 10.38%)	(0.001, 17.31%)
Hidden Size	(32, 19.23%)	(64, 17.31%)	(128, 18.33%)

transformations to increase data diversity and volume. The second direction was implementing a frame-based CNN classifier as a baseline to evaluate how much spatial information alone contributes to sign recognition. Instead of using 15 frames per video, we selected the middle frame, which we believe would most likely contain critical information, and passed it through a CNN layer that extracts local spatial features by applying learnable filters across the image. The resulting accuracy was 32.64%, which, while still relatively low, outperformed the 19.23% accuracy of our CNN-RNN model.

This reveals several insights. First, it shows that the signs are at least partially recognizable from spatial features alone. Second, it suggests that the RNN model may actually be harming performance, possibly due to the absence of hands in some frames, or the noisy background and presence of human faces misleading the model into learning irrelevant patterns and overfitting to noise. This suggests we are likely better off focusing on improving spatial clarity before further pursuing temporal modeling. As a potential fix, we are implementing SAM2 and MediaPipe to mask out the hands, as discussed in the next section. Once they are applied, the features should be easier to capture, and we also plan to reduce the complexity of our model to cut training time.

3.3 SAM2 + MediaPipe

3.3.1 SAM2

The main purpose of implementing SAM2 is to preprocess the image, extract the binary mask, and use it as input for the proposed model. Masking is effective in defining the region of interest (ROI). As shown in Figure 2, a pixel value of 0 (black) represents the background, and a pixel value of 1 (white) represents the ROI. Masking facilitates feature extraction by capturing meaningful information from the image. Additionally, it enhances computational efficiency by providing an extra layer of information, allowing the model to focus on the ROI rather than performing unnecessary computations on the background.

Figure 2: Masked image and its binary mask representation.

3.3.2 MediaPipe

The MediaPipe Hand Landmarker task can detect the keypoint localization of 21 hand-knuckle coordinates (2). This solution can be used to locate and visualize hand keypoints. It outputs hand landmarks in relative coordinates, as well as the handedness (left or right hand) of multiple detected hands (2). Specifically, MediaPipe's handedness feature is used to determine whether each detected hand is left or right, draw a bounding box around it, and input it into SAM2.

4 Experiments

4.1 SAM2

4.1.1 Experimental Setup

All experiments were conducted in Google Colab using a T4 GPU with PyTorch version 2.5.1 and Torchvision version 0.20.1. The *sam2.1_hiera_large.pt* checkpoint was used. According to the SAM2 documentation, Google Colab A100 or L4 GPUs are recommended, and Python $\geq$ 3.10, PyTorch $\geq$ 2.5.1, and Torchvision $\geq$ 0.20.1 are required.

4.1.2 Automatic Mask Generator

The automatic mask generator is provided through the *'SAM2AutomaticMaskGenerator'* class. It returns a list of masks, where each mask is a dictionary containing metadata about the mask, including segmentation, area, bbox (bounding box), and predicted_iou. There are also hyperparameters, such as 'points_per_batch', that can be tuned to control the density of sampled points and the thresholds for removing low-quality masks.

The results of the automatic mask generator are shown in Figure 3. Unfortunately, the automatic mask generator was not suitable for extracting only the features of the hands. While it performed well in distinguishing humans from the background, it did not effectively isolate specific body features. Instead, it often masked the human figure as a whole.

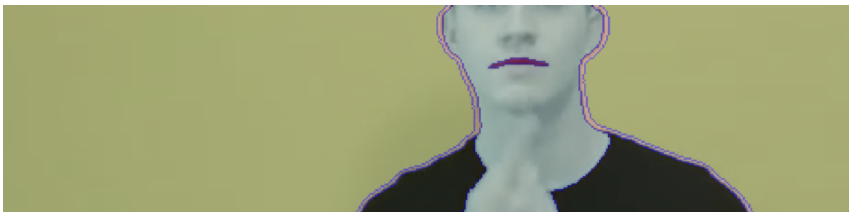

Figure 3: Resulting mask generated by the automatic mask generator.

4.1.3 Video Predictor

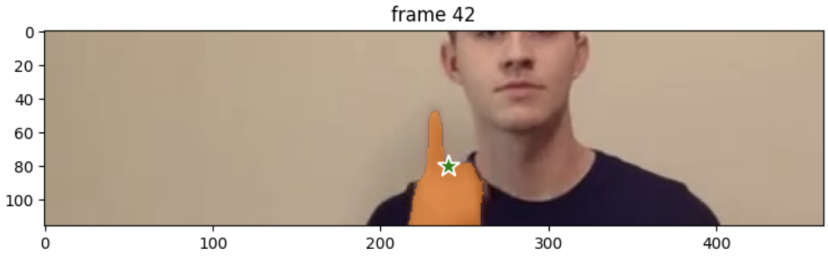

Figure 4: A positive region prompt was provided in the initial frame.

SAM2 also provides a video predictor, which allows the user to prompt the initial object and generate a masklet for the entire video. The prompt enables the user to precisely mask the targeted object. It can be provided in the form of point coordinates or bounding box coordinates.

To generate the masklet throughout the entire video, the inference state for interactive video segmentation must first be initialized. After initialization, use the propagate_in_video API, which propagates the prompts across frames.

The result of the experiment is shown in Figure 4, where a positive prompt was provided to mark a region in the initial frame. After running the propagate_in_video API, the masklet

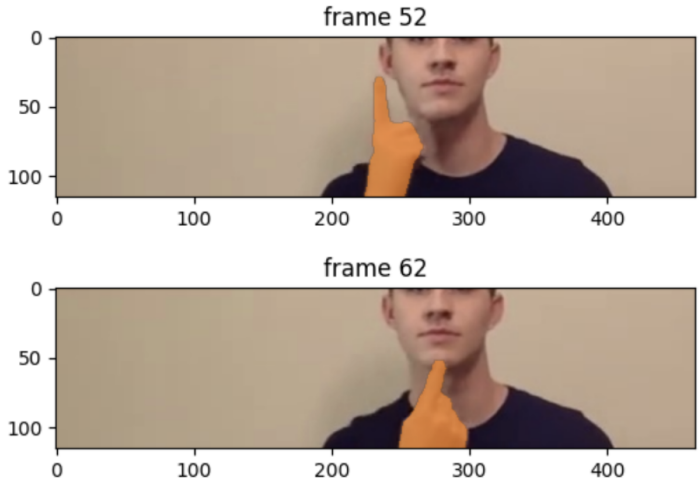

Figure 5: SAM2 generated a masklet over the entire frame.

was successfully generated for the entire video, effectively masking only the targeted hand with remarkable accuracy [Figure 5]. Notably, even when the hand temporarily moved out of the frame, the predictor successfully resumed masking the hand once it reappeared.

The downside of using the video predictor is that manual prompting is required for the entire dataset. This process may involve editing the frames to identify the initial frame where the hand object appears and manually entering the (x, y) coordinates of the hand object.

4.1.4 SAM2 Video Predictor + MediaPipe

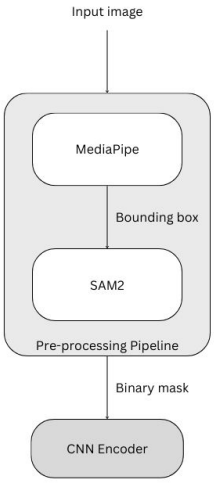

Figure 6: Pre-processing pipeline.

To mitigate this issue, we incorporated MediaPipe into our preprocessing pipeline. The overall process and experimental results using MediaPipe and the SAM2 video predictor are shown in Figure 6. Since SAM2 accepts bounding boxes as a prompting input format, we first detect handedness features to identify left and/or right hands. Note that the SAM2 masklet needs to be initialized when the object is within the frame, so we begin from the index of the frame where a hand is first detected, and this is determined by checking that the MediaPipe output is non-empty. We then draw bounding

boxes around the detected hands and input them into the SAM2 video predictor. The top-right corner image from Figure 7 shows the bounding box input and the corresponding masklet generated in the initial frame to initiate tracking throughout the video. As shown in the two bottom images from Figure 7, the masklet was successfully generated for the entire video, effectively masking only the target without requiring any manual input from the user.

Jupyter notebook file shows the results in detail : `https://github.com/mmiachang/ece324/blob/main/notebooks/SAM2%2BMediaPipe-automasking.ipynb`.

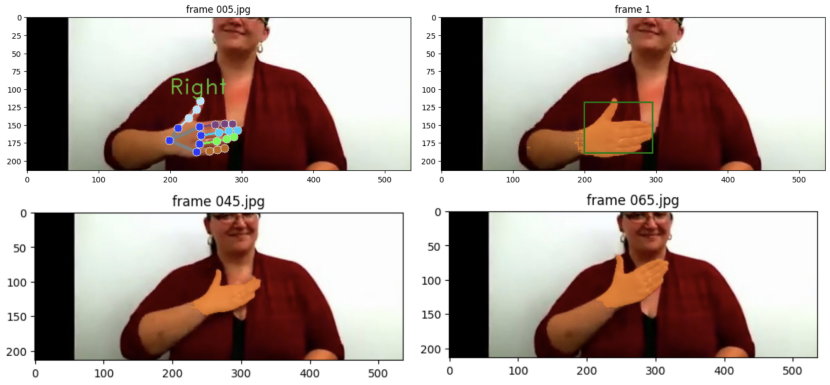

Figure 7: Results of the automated pre-processing pipeline. The top-left corner shows the visualization using the MediaPipe library, while the top-right corner displays the bounding box and the mask generated over the initial frame. The two bottom images show the generated masklet applied over the entire frame.

However, there are still potential improvements that can be made to seamlessly integrate the masking approach with the CNN model.

First, as shown in Figure 7, while using a bounding box as a prompt input is generally effective, the autogenerated masks are occasionally less accurate [Figure 8] compared to those generated using point coordinate inputs, as illustrated in Figure 4.

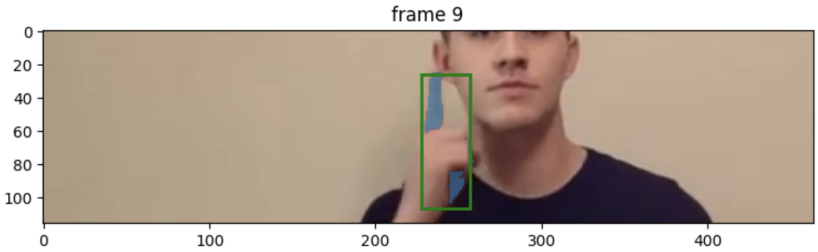

Figure 8: Less accurate masklet generated in the initial frame.

Second, our automation code initiates the process using the first frame in which a hand is detected. However, this approach can sometimes misidentify the ideal starting frame. For instance, in Figure 9, the desired initial frame is the one containing both the left and right hands, but the system begins processing as soon as either hand is detected. This can lead to inaccurate mask initialization, which subsequently affects the quality of the entire masklet sequence.
Lastly, improved computational resources are needed to efficiently generate the masklets. For example, producing the masklets for the three demo videos used in Figures 7, 8, and 9 - with 12, 22, and 25 frames respectively - took 12:24, 27:06, and 11:08 minutes using a Google Colab T4 GPU. This level of processing time is inefficient and will scale poorly as the dataset size increases. These three limitations are identified as areas for future improvement.

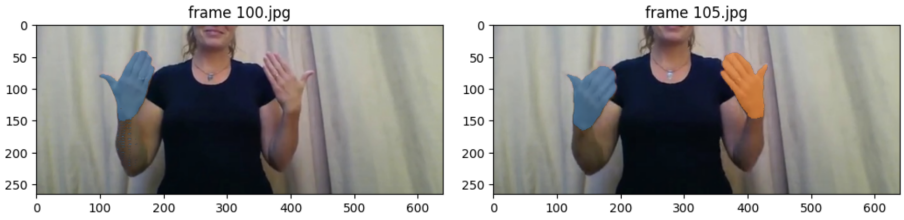

Figure 9: Incorrect initial frame detecting a single hand instead of two.

5 Conclusion

To conclude, the highest testing accuracy achieved so far is 19.23%, indicating significant room for improvement. Through two directions of investigation, we identified key limitations in our current approach. First, increasing the amount of training data demonstrated a positive correlation between dataset size and accuracy, highlighting the need for data augmentation techniques such as rotation, translation, and other transformations to expand the dataset. Second, implementing a single-frame CNN yielded a higher accuracy of 32.64%, suggesting that the RNN component is not effectively contributing to performance. This result emphasizes the importance of improving the preprocessing pipeline to enhance spatial feature extraction, which appears to be more critical at this stage of the model's development.

To address this, we explored the potential of leveraging SAM2 to generate binary masks that isolate relevant regions, such as hands, thereby improving feature extraction and training efficiency. We automated the interactive segmentation of SAM2 by inputting the output of the MediaPipe library. However, there are still potential improvements for occasional low accuracy, initial frame identification, and poor computational resources.

Therefore, the low testing accuracy could potentially be improved through SAM2 automation and by training our model with masked values, but due to time constraints, this will be a potential future task.

References

[1] Ravi, N., Gabeur, V., Hu, Y.T., et al. (2024). *SAM 2: Segment Anything in Images and Videos*. arXiv preprint arXiv:2408.00714.

[2] Lugaresi, C., Tang, J., Nash, H., et al. (2019). *MediaPipe: A Framework for Building Perception Pipelines*. arXiv:1906.08172.

[3] MS-ASL. Microsoft Research. (2019, August 5). https://www.microsoft.com/en-us/research/project/ms-asl/

Acknowledgment

A generative AI tool (ChatGPT) was used to check the grammar of the report. However, the original content was fully written by the team members themselves.