Minnesota State University, Mankato

Group Project Report
Detecting American Sign Language

CIS630 Advanced Artificial Intelligence Systems

TEAM 2

EunSoo Park
Sarthak Dhakal
Wengel Tsegaselassie

Table of Contents

Introduction

As our society becomes increasingly diverse, the importance of effective communication continues to grow. This shift underscores the need for innovative solutions that can bridge language barriers and promote inclusivity. American Sign Language (ASL) plays a crucial role as a primary mode of communication for many within the deaf and hard-of-hearing communities. Yet, the limited availability of accessible translation tools still presents significant challenges for seamless interaction between ASL users and the wider public.

This report explores how Artificial Intelligence (AI) can be applied to enhance automated ASL translation. The system we propose is designed to recognize hand gestures from images, convert them into text, and — as a longer-term goal — generate spoken language from the text output. By leveraging advanced AI techniques, our vision is to create a tool that not only supports communication for ASL users but also helps foster greater inclusivity and understanding across different social settings.

In this study, we will examine the technological feasibility, explore potential methods, and discuss the broader implications of implementing such a system. Through our findings, we aim to contribute to the growing conversation around accessible communication technologies and empower individuals who rely on sign language in their daily lives.

Data

1. Data Instruction and Normalization

To prepare our American Sign Language (ASL) image dataset for effective machine learning model training, I implemented a data normalization pipeline.
The raw images, as initially provided, were in color and had varying pixel intensities, which could hinder the model's ability to learn consistent patterns.

The normalization process consisted of two key steps:
Pixel Normalization: first, we normalized the pixel values. Grayscale pixel values typically range from 0 to 255. By dividing each pixel value by 255, we scaled them to a range of 0 to 1. This ensures that all pixel intensities are on a uniform scale, preventing any single pixel from disproportionately influencing the model. (As shown in this code snippet: bw_array=np.array(bw_img)/255.0)
Black and White Conversion: Then, we converted the images to black and white (binarized) since our UI was based on grey scale. This step further simplified the images by using a threshold to distinguish between foreground and background. This highlights the essential features of the hand gestures, making it easier for the model to identify and learn them.

Throughout this process, we maintained the original directory structure of the dataset. This ensures that the organized data remains in a format that is easy to manage and use for subsequent model training. These normalization steps are crucial for improving the accuracy and efficiency of our ASL recognition model

2. Data Structure and label assignment

We obtained our dataset from Kaggle, which consists of labeled images organized into directories based on their respective classes. Each class represents a character from the alphanumeric set, including all uppercase letters (A–Z) and digits (0–9), resulting in a total of 36 distinct categories. For instance, all images corresponding to the character "A" are stored in a folder named "A", while images of the digit "4" are located in a folder named "4".

To load and preprocess the dataset, we utilized the ImageDataGenerator function, which not only handles image augmentation and batching but also automatically assigns labels to each image based on the name of its parent directory. This streamlined the labeling process by mapping each image to its corresponding alphanumeric class.

Model

1. Model Architecture

We adopted a transfer learning approach using the MobileNetV2 architecture as the base model for feature extraction, initialized with weights from ImageNet.

The top classification layers were removed, as they were tailored to ImageNet's object classes and not applicable to our 36-class sign language dataset. We initially froze the earlier layers of the model to preserve their general image feature extraction capabilities and added custom fully connected dense layers on top. These layers were responsible for learning to associate the extracted features with our specific sign language classes.

Our initial model achieved a validation accuracy of approximately 63%. To improve this, we unfroze the last 50 layers of the MobileNetV2 base and fine-tuned them. This helped the model adapt more specifically to the sign language dataset, increasing accuracy to 66%. However, a significant gap between training (80%) and validation accuracy indicated overfitting.

2. Model Training

We used the Adam optimizer (Adaptive Moment Estimation) to compute gradients and update weights. Initial training was conducted for 15 epochs, but we later adjusted our strategy to train for a larger number of epochs while addressing overfitting through:

- Early stopping, to halt training when validation performance stops improving.
- ReduceLROnPlateau, to decrease the learning rate dynamically and aid convergence.

To further improve generalization, we introduced additional regularization techniques:

- L2 regularization on dense layers to penalize large weights.
- Label smoothing to soften ground truth labels and reduce overconfidence in predictions.

Additionally, we used MediaPipe to crop out only the hands from the images (region of interest) before feeding them into the model. This helped eliminate background noise and focus on relevant features.

After these adjustments, the model's validation accuracy increased significantly to 84%. We also experimented with alternative architectures such as EfficientNet and ResNet, but MobileNetV2 consistently outperformed them in our specific use case.

User Interface

1. Client Side:

For the facilitation of real-time ASL detection in an accessible and user-friendly manner, we decided to move ahead with the development of a web application that allows interaction with the prediction model for sign detection. The UI has been developed to be minimalistic yet functional, giving the users the ability to use their existing device and not needing to install any additional software tools to run the program.

The getUserMedia() API from JavaScript is used to get access to the user's webcam, allowing the UI to capture real-time video frames from the user's hand gestures. The UI has a button labeled "Predict" the current video frame is captured, and the image is then converted into base64 encoded data and sent to the backend using AJAX POST request.

2. Server Side:

In the server side, before the image is passed to the trained MobileNetV2Model, the preprocessing is handled by MediaPipe Hands, a library from Google for advanced hand tracking. It detects 21 hand landmarks within the image and then, using these landmarks, a tight bounding box around the hand is created, which removes any unnecessary details and elements that would affect the prediction from the model. After the hand area is cropped, the image is resized to match the 224x224 pixels as expected by the model. Additionally, normalization of the image is also done by scaling the pixel values of the image from [0,255] to [0,1].

The model then takes the input and instantly provides a prediction, identifying English alphabets and numbers from 0 to 9. The predicted value is then also converted into audio using the Google-Text-to-Speech (gTTS) library, which provides us with an audio file.

Finally, the entire system, consisting of the trained model, backend, and the frontend has been containerized to ensure that deployment across multiple environments doesn't affect the reproducibility of the system. The result is displayed as a text on the UI and also plays the audio to the predicted output, as shown in the demonstration of the system.

Result

Our initial model, built using MobileNetV2 with minimal adjustments, achieved a validation accuracy of 63%, indicating room for improvement. After several iterations and tuning efforts, we saw a moderate increase to 66%, though overfitting remained a concern given the higher training accuracy.

With further enhancements—including regularization techniques and improved data preprocessing—the model’s validation accuracy improved significantly to 84%. This 21% increase from our starting point demonstrates clear progress in the model’s ability to generalize. Because the highest accuracy we were able to achieve was 84%, we have incorrect translations once in a while. Here's an example of some of the translations below.

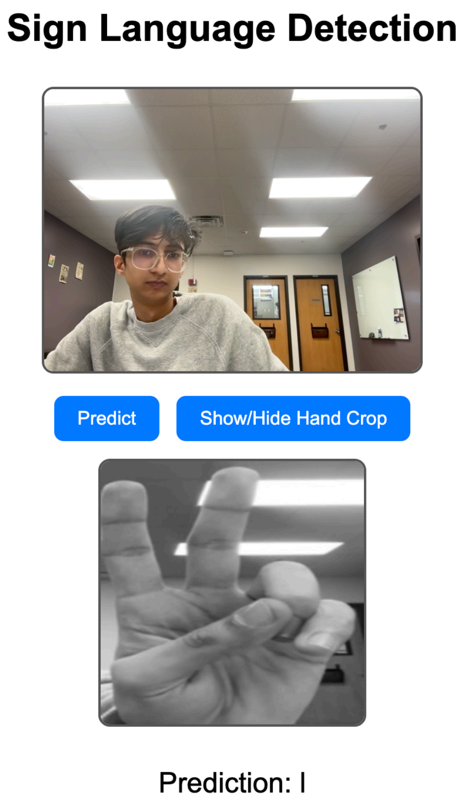
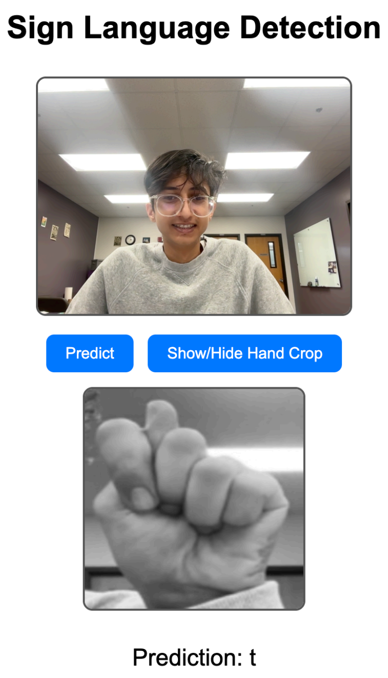
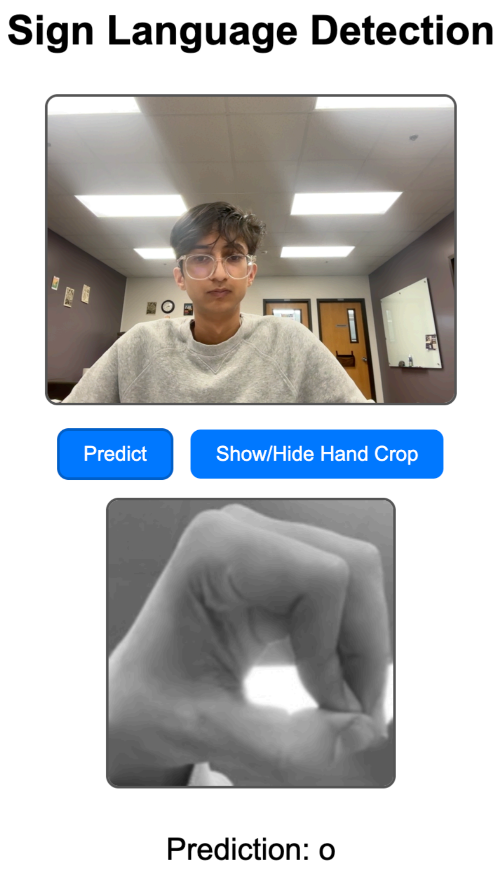


Figure 1. Translation of the number 2 is labelled as I. This is an example of a wrong prediction.


Sign Language Detection

Prediction: t

Figure 2: Translation of the letter t. This is an example of a prediction that works


Sign Language Detection

Prediction: o

Figure 3: Translation for letter O. This is also another example of a correct prediction.

Challenges

Throughout the development and training of our sign language recognition model, we encountered several key challenges that shaped our approach and informed our learning process.

1. Long Training Times

One of the most significant challenges we faced was the length of training time, especially when experimenting with larger models like ResNet and EfficientNet. In some cases, training took over four hours, which limited the number of experiments we could perform in a single day. This constrained our ability to fine-tune hyperparameters and test additional regularization techniques in a timely manner.
We later realized that our campus may offer more powerful computing resources, which could have drastically improved our workflow. However, we only discovered this late in the process. In hindsight, proactively exploring available infrastructure earlier would have enabled us to train faster and iterate more frequently.

2. Overfitting

Overfitting was another persistent issue during training. Although our initial model achieved around 80% training accuracy, the validation accuracy lagged behind at 63–66%, indicating that the model was not generalizing well to unseen data. This prompted a shift in our strategy, leading us to apply techniques such as early stopping, learning rate scheduling, L2 regularization, and label smoothing, which helped improve validation performance.

3. Data Quality and Preprocessing

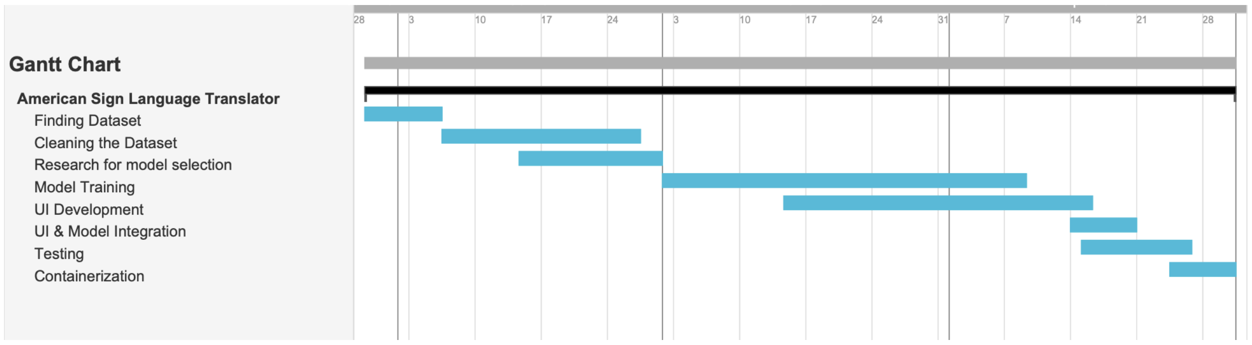
A critical turning point in our project was the realization that data quality and preprocessing are just as important as the model architecture itself. At one stage, we were adjusting model parameters in hopes of improving predictions, only to later discover that we had been using an incorrect version of our normalized dataset. This oversight consumed valuable time and highlighted the need to verify data pipelines before training.
To further improve input quality, we used MediaPipe to isolate and crop the hand region in each image, effectively reducing background noise and allowing the model to focus on the region of interest.

4. UI Integration Challenges

Integration of the user interface with the real-time webcam based prediction web application had introduced challenges, primarily with dealing with the MediaPipe Hands to detect and crop the region of interest using as variable conditions where the web application can be used causes issues with not identifying the hand when in poorly lit areas, hand being tilted and out of frame resulting in incorrect cropped pictures leading to it identifying no hands.

To mitigate the issues, we implemented the padding for the bounding box to ensure that the hand remains fully captured within the frame and also updated the code to dynamically calculate and adjust the bounding box based on the area the hand landmarks are spread on the frame. These improvements helped us reduce faulty crops and improved model's reliability during use.

Timeline and Contribution



- **EunSoo Park - Data Processing Lead**
- **Sarthak Dhakal - Deployment/InterfaceLead**
- **Wengel Tsegaselassie - Model Development Lead**

Here is the Gantt chart outlining the timeline for our project. We begin with Finding the Dataset, followed by Cleaning the Dataset. This process was leaded by me. During this time, we also conduct Research for Model Selection to determine the best approach for recognizing ASL gestures.
And then we move into Model Training, which Wengel lead the process. Parallel to this, Sarthak started developing UI, overlaping with UI and Model Integration.
After model training, we begin the Testing phase to evaluate system performance and ensure accuracy.
Finally, we wrap up with Containerization, preparing the system for deployment and easy scalability.
This timeline shows a structured, step-by-step progression, with some overlapping tasks to keep the project moving efficiently toward completion by the end of April.