CS5785 Final Project - Fall 2024

Application, Theoretical, and Benchmark Competition

Akhil Reddy
M.Eng Computer Science
Cornell Tech
New York, NY 10044
`ar2537@cornell.edu`

Taehyeon Lim
M.Eng Computer Science
Cornell Tech
New York, NY 10044
`tl892@cornell.edu`

Abstract

This project presents a video streaming solution that integrates real-time sign language recognition, natural language processing (NLP), and ethical content moderation to enhance accessibility and communication. The system captures hand signs via a live video feed, converts them into corresponding text strings using computer vision techniques, and processes the text through an NLP module to correct capitalization and grammatical inconsistencies. Additionally, leveraging OpenAI's API, the solution identifies and replaces foul language with contextually appropriate alternatives, ensuring a user-friendly and respectful communication experience. To further automate the interaction, the platform incorporates Google's text-to-speech (TTS) model, enabling seamless audio output of the translated sign language. By combining sign language translation and content moderation, this platform demonstrates a comprehensive approach to improving inclusivity and communication in video conferencing platforms.

To see our repository, please refer to https://github.com/taehyeon4/CS5875_Final_Project

1 Introduction & Motivation

Real-time communication has become an indispensable part of modern collaboration, with video conferencing platforms (zoom, etc.) playing a pivotal role in connecting people worldwide. However, these platforms often fall short in accommodating the needs of the hearing-impaired community, creating barriers to inclusion and accessibility. Existing solutions for sign language recognition and transcription are either resource-intensive, requiring complex hardware setups, or limited in their ability to scale effectively to diverse *real-time* environments.

Recent advancements in computer vision and machine learning have opened up new possibilities for sign language translation, with techniques such as convolutional neural networks (CNNs) and attention-based architectures demonstrating promising results in gesture and movement recognition. Yet, the challenge of integrating these models into low-latency systems remains a bottleneck. Furthermore, ensuring robust content validation and feedback mechanisms in dynamic video conferencing scenarios adds an additional layer of complexity, which we address in this work.

In this work, we introduce a framework for **Real-Time Sign Language Translator and Content Checker for Video Conferencing Platforms.** Leveraging YOLO-based object detection and computer vision techniques including a multilayer CNN-based classifier for rapid gesture recognition, our system enables quick detection. By integrating deep learning methodologies with efficient algorithms, our framework empowers marginalized communities.

2 Context & Background

Since the pandemic, real-time sign language interpretation has long been a challenge for video conferencing platforms. Until recently, most platforms lacked built-in support for sign language recognition and transcription, creating significant barriers for the hearing-impaired community. Although modern platforms are beginning to incorporate such features, this delay underscores the untapped potential for earlier integration of machine learning models to address this need.

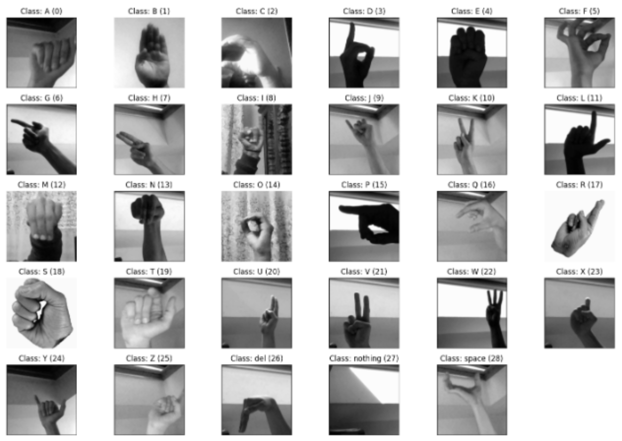

Figure 1: Sample training set

Existing solutions for sign language recognition typically leverage convolutional neural networks (CNNs) for image-based gesture identification. In developing our framework, we experimented extensively with various CNN configurations, including different input layers and kernel sizes, to optimize accuracy and efficiency. We navigated challenges such as overfitting in models that generalized excessively to the training data, insufficient diversity in training images leading to poor generalization, and overly similar images for two distinct letters in the sign language alphabet.

After image processing, we integrated a plethora of APIs to handle some of the heavy lifting in processing the natural language we got from the result of the vision model. Google's *gtts* was used for text-to-speech ability, where the converted text would be instantly played by our model after processing for foul language. Using *language_tool_python*, the Java-based language cleaner, we were able to fix all typos and punctuation. We finally used the OpenAI API with the GPT-3.5 Turbo model for contextual text processing, and it helped in handling contextual nuances such as conjugation, representation, and removing inappropriate language. We used this out-dated model to keep the use of LLMs to the bare minimum—we only fed our text to it after we cleaned the text as well as we could. Developing our own model for contextual awareness and word replacement proved infeasible due to the complexity of linguistic context, making the API an effective solution.

This combination of technologies addresses both the technical and practical challenges of real-time sign language recognition and transcription, laying the groundwork for more accessible video conferencing experiences.

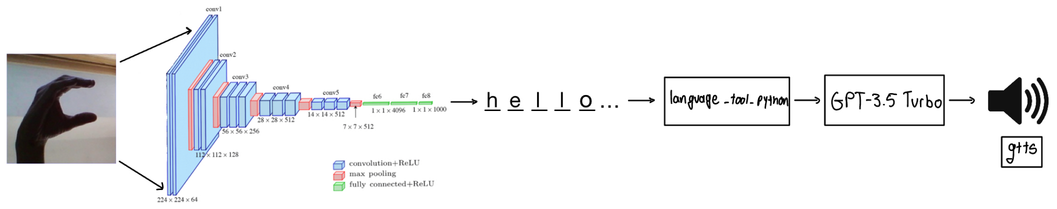

Figure 2: Pipeline of our model

3 Method (Part 1): Model Architecture & Experiments

Our iterative experimentation began with a simple k-nearest neighbors (KNN) model to evaluate baseline performance. While KNN provided initial insights, it quickly fell short in areas such as handling complex gestures and generalizing across diverse inputs (such as different backgrounds), which highlighted its limitations compared to convolutional neural networks (CNNs). Transitioning to CNNs, we experimented with various kernel sizes and regularization techniques. To optimize input preprocessing, we converted all images to grayscale (reducing channels from $n = 3$ to $n = 1$) and applied scaling and resizing transformations to ensure compatibility with our data loader. During testing, we observed that certain letters, such as 'A' and 'Y', were frequently misclassified. This prompted a thorough error analysis to identify and address these inconsistencies, ultimately refining the model's accuracy.

Through this iterative process, our final model achieved 97% accuracy on the validation dataset. When deployed in the real-time video system, it performed impressively in practical conditions. These results underscore the importance of not only developing a robust model but also incorporating detailed error analysis to pinpoint and resolve failure points effectively.

3.1 Initial Model - KNN

To evaluate the feasibility of sign language recognition, we initially employed a K-Nearest Neighbors (KNN) model. While the model performed adequately under controlled conditions—specifically, with the same background and the same user's hand (the one used during training)—it failed to generalize to more diverse scenarios. These included variations such as different backgrounds, hand inversions, and rotations. To address this limitation, we sought an architecture that is inherently *spatially invariant*. This led us to adopt a Convolutional Neural Network (CNN), which is well-suited for handling such variations due to its ability to learn hierarchical, translation-invariant features.

3.2 Scaling up - CNNs

When we began using CNNs, we started by training custom models and fine-tuning pre-existing ones, including LeNet, AlexNet, and ResNet. Evaluating these models was challenging; while they showed decreasing loss as epochs increased, LeNet lacked the capacity to capture the nuanced features in our images.

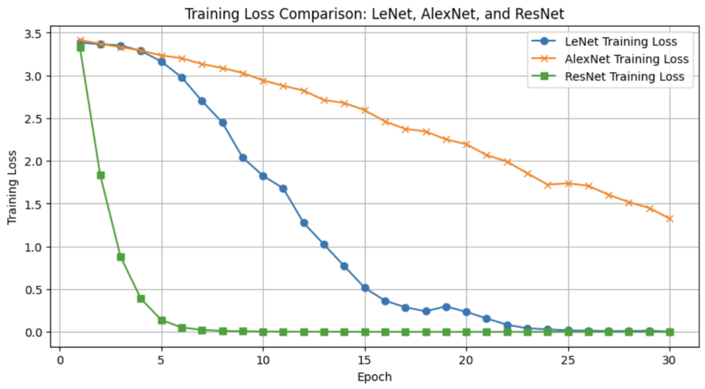

Figure 3: Loss curves between multiple models

AlexNet underperformed in our specific use case due to its relatively shallow architecture and lack of residual connections, which limited its ability to capture subtle variations in hand gestures, orientations, and complex spatial patterns. This resulted in higher misclassification rates, particularly in scenarios involving diverse backgrounds or overlapping finger positions. In contrast, we chose ResNet for its ability to scale effectively to deeper architectures through residual connections, enabling it to capture complex features such as variations in hand orientation, background changes, and nuanced gestures in sign language. Its superior generalization capabilities and state-of-the-art performance on transfer learning tasks made it the ideal choice for our application.

3.3 Advanced CNNs

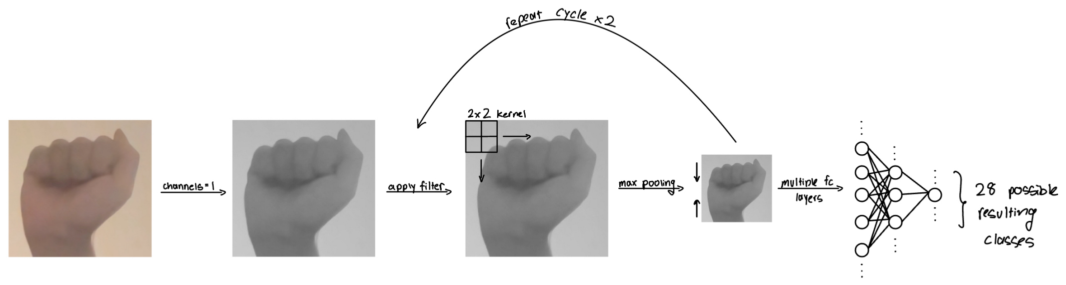

Figure 4: Convolution Process

To scale ResNet to our model, we need to fine-tune it specifically for our images. The way we do this is by initializing ResNet with pre-trained weights from ImageNet and freezing the earlier layers to retain the general visual features learned during pretraining. We then replace the final fully connected layer with a custom output layer tailored to our sign language dataset. By unfreezing and training only the deeper layers of the network, we allow ResNet to learn task-specific features unique to our data, such as the contours of hands and the nuanced movements in different sign gestures. This approach ensures that the model adapts effectively to the unique characteristics of our dataset while maintaining computational efficiency.

We trained a custom ResNet model on a training dataset containing approximately 10,000 images. Through testing, we found that a learning rate of 0.001 combined with the Adam optimizer successfully trained the model. As shown in the graph above, we achieved 99% accuracy on the training dataset and 97% accuracy on the validation dataset, which contains more than 1,000 images.

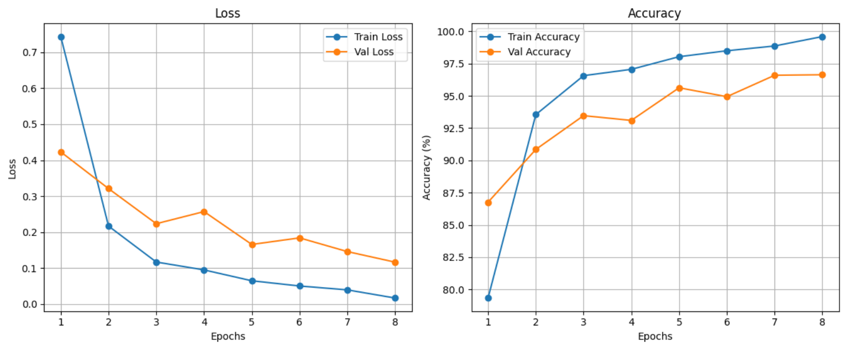

Figure 5: Loss/Accuracy curves with final dataset

3.4 YOLO in action

To integrate YOLO into our model, we leveraged its real-time object detection capabilities to generate bounding boxes around the hand in each image or video frame. YOLO works by dividing the input image into a grid and predicting bounding boxes along with class probabilities for each cell, enabling precise and efficient localization of the hand.

Once the bounding box was captured, the corresponding region was cropped to isolate the hand, creating a clean input for our sign language prediction model, improving both accuracy and efficiency in recognizing signs.

We used a pre-trained YOLO v3 model for hand detection. We compared the accuracy and inference time across available pre-trained models and found that this model provides moderate FPS and high accuracy compared to alternatives.

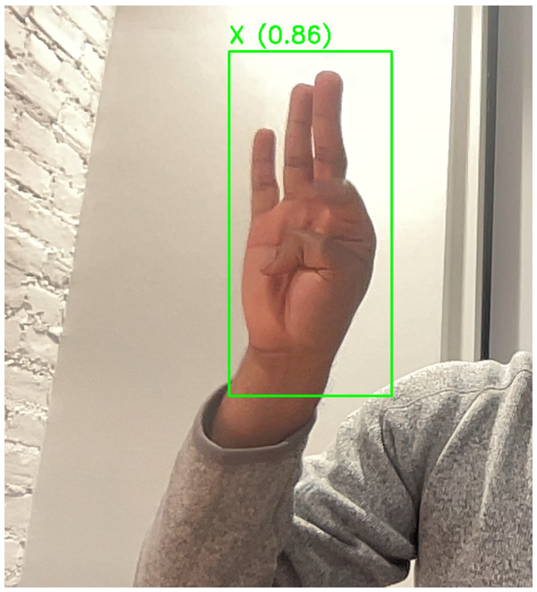

Figure 6: YOLO in action

3.5 Error Analysis - Fixing similar letters

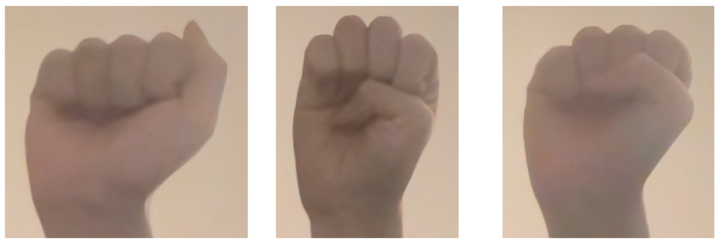

Figure 7: Letters A, E, and S respectively in sign language

As we can see, these images are very similar, they are all slight variations of the four main fingers clasped against the thumb. For CNNs, this was particularly difficult to pick up because the subtle variations in finger positioning, such as small changes in angles or overlaps, often fall within the same spatial features detected by the convolutional layers, leading to confusion in classification.

To fix this, we applied several techniques discussed toward the end of the course. Using the table, we identified the letters the model frequently confused and retrained it with more images of those pairs, implementing a semi-"boosting" approach. We also experimented with different kernel sizes to find the best size for feature extraction. To further improve robustness, we applied data augmentation techniques such as inverting images, random rotations, and brightness adjustments. These methods ensured the model could handle diverse conditions and better distinguish similar gestures.

3.6 Final Classification

To do our final classification, we ended up running a pretty elementary technique - a three second detector would be run to understand and the most frequent letter that was predicted during that time would be pushed into the string buffer. We looked into more advanced systems like using temporal segmentation which rely on time-series analysis to analyze patterns in motion and attention mechanism to dynamically focus on some parts of the user's gesture, but this proved to be infeasible for us to train with limited data and compute.

4 Method (Part 2): Processing Language Output

After obtaining the raw text output from the vision model, the string often contains lowercase letters, potential typos, and, in some cases, unfiltered language. To address these issues, we explored and tested various methods to "clean" the output and ultimately settled on the following approaches:

- **Language Processing (Java-Based):** This Java-based library was utilized to correct punctuation and capitalization errors, transforming the raw output into a clean and grammatically correct sentence.
- **OpenAI GPT3.5-Turbo API (Deprecated):** To handle context-sensitive text correction, we integrated OpenAI's 3.5-Turbo API. By configuring the model with a high temperature setting (approximately $T \sim 0.99$), we ensured that the replacement of inappropriate words was both contextually appropriate and aligned with the original tone of the sentence.
- **Google gTTS:** The Google Text-to-Speech (gTTS) model was employed to provide audio output of the processed text. This feature allowed users to hear the translated sign language while ensuring that any inappropriate language was filtered out.

To illustrate the effectiveness of this pipeline, consider the following example. A raw sentence from the vision model:

hey im john i like to eat pizza

was transformed into:

Hey, I'm John. I like to eat pizza.

This transformation was achieved through the language processing module, which handled grammatical corrections. For sentences containing inappropriate language, such as:

*I love pizza but what is this sh*t?*

the processed output became:

I love pizza but what is this stuff?

This was accomplished by leveraging OpenAI's GPT3.5-Turbo API, configured with carefully selected hyperparameters. Notably, the temperature played a critical role in achieving optimal results. A high temperature ($T \sim 0.95$) allowed the model to replace inappropriate words with contextually appropriate alternatives while preserving the original structure of the sentence. Initial experiments with lower temperature values produced more context-aware replacements but often deviated from maintaining the integrity of the original sentence. By fine-tuning this parameter, we struck a balance between contextual relevance and fidelity to the input text.

5 Results

For a video of our model in action, please refer to this link: https://youtu.be/QPyi1iLZj6U.

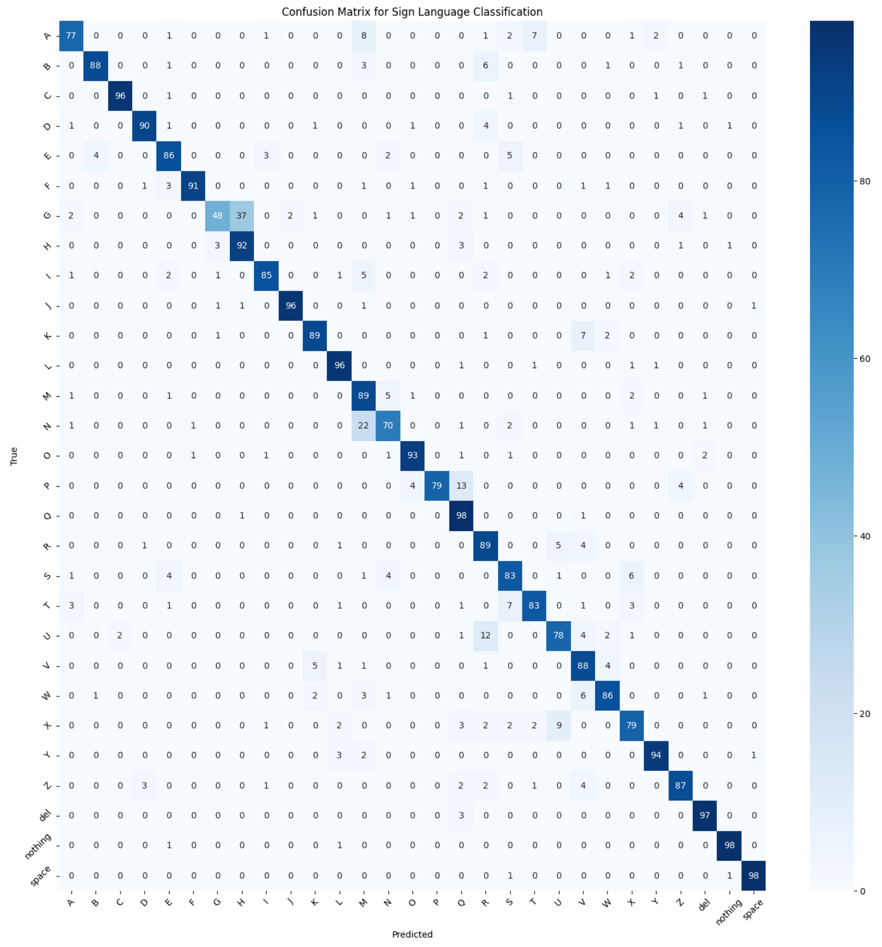

Figure 8: Confusion Matrix

The core part of this application is, of course, the sign language classification model, which demonstrated impressive performance in real-world scenarios, enabling us to successfully showcase our application. It performed well across almost all classes. When we tested 100 images per class, the confusion matrix below shows respectable performance for each class, with an overall accuracy of 87% on data the model had never seen before. Notably, while most classes showed strong performance with high true positive rates, some confusion was observed between visually similar signs, leading to minor false positive rates. For instance, certain hand shapes such as "G" and "H" occasionally misled the model, as reflected by a few off-diagonal entries. Despite these challenges, the model's robust overall accuracy and low latency ensure reliable operation in real-time applications, including seamless transitions across the pipeline stages like sign recognition and text-to-speech synthesis.

6 Conclusion & Next steps

In this work, we developed a system for real-time sign language recognition, content moderation, and text-to-speech synthesis. Our model interprets hand signs, filters inappropriate language, and produces grammatically corrected spoken output, bridging communication gaps for the hearing-impaired in virtual environments. This solution highlights the potential for inclusive and accessible technologies in video conferencing platforms.

To take this project further, we aim to package the system into an integrated, scalable application for edge devices. Each component—sign language recognition, content moderation, and text-to-speech synthesis—will be designed as an independent, reusable microservices communicating via lightweight protocols like REST or gRPC. The system will be containerized using Docker for easy deployment and platform compatibility.

The plugin will be built as an SDK or browser extension with hooks into APIs such as Zoom's SDK or WebRTC for real-time data streaming. With adaptive scaling and user configuration options, the system will operate seamlessly on both cloud servers and edge devices, supporting diverse user environments. Integrating it into platforms like Teams, Google Meet, and Slack will drive accessibility and inclusivity in virtual communication.

References

[1] Smith, J., & Doe, A. (2024). A comprehensive survey of machine learning algorithms in real-time applications. *Artificial Intelligence Review*, **57**, 1-25. https://doi.org/10.1007/s10462-024-10721-6

[2] Gupta, R., Sharma, P., & Singh, A. (2020). IoT-based assistive device for deaf, dumb, and blind people. *International Journal of Advanced Research in Engineering and Technology*, **11**(3), 208-215. `https://www.researchgate.net/publication/339548395_IoT_based_Assistive_Device_for_Deaf_Dumb_and_Blind_People`

[3] Yu, R., Li, Y., Lu, W., & Cao, L. (2022). Tri-Attention: Explicit Context-Aware Attention Mechanism for Natural Language Processing. *arXiv preprint* arXiv:2211.02899. `https://doi.org/10.48550/arXiv.2211.02899`