Real-Time Sign Language Translation Using Deep Learning

Project Milestone Outline

Team Members: Akhil Reddy, Taehyeon Lim

NetIDs: ar2537, tl892

Project Category: Application, Theoretical, or Benchmark Competition

GitHub Repository: https://github.com/taehyeon4/CS5875_Final_Project/tree/main?tab=readme-ov-file

Colab Notebook: https://colab.research.google.com/github/taehyeon4/CS5875_Final_Project/blob/main/Main.ipynb

---

Motivation

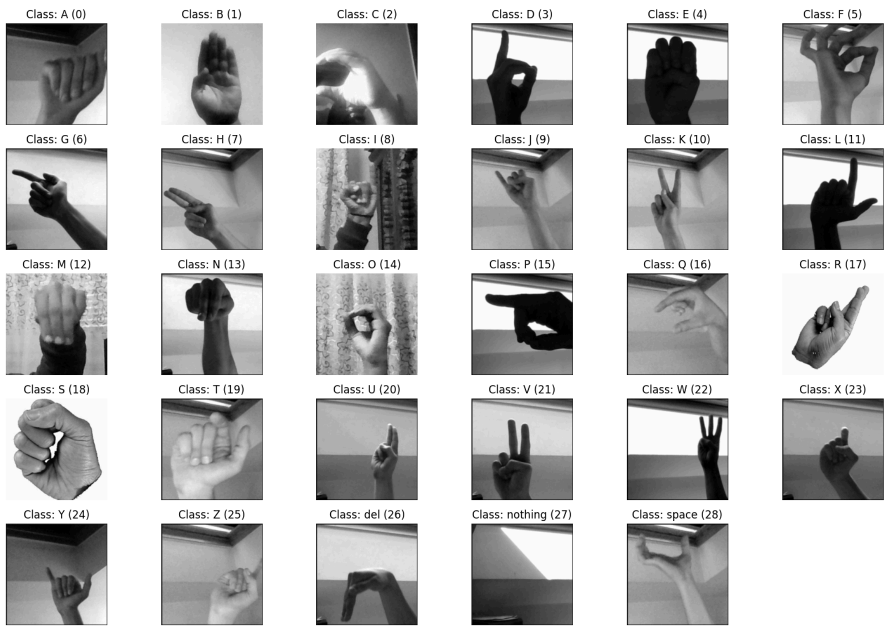
Problem Statement:
This project addresses the challenge of real-time sign language translation in video conferencing platforms, which can significantly enhance accessibility for deaf and hard-of-hearing individuals. In the digital era, where virtual meetings are prevalent, accessibility remains a critical issue, especially for sign language users who are often excluded from full participation. Existing solutions are either too slow or lack accuracy, especially in real-time applications.

Our approach leverages machine learning, specifically deep learning and computer vision, to develop a system capable of identifying and translating sign language gestures as they happen. This problem is particularly challenging due to the complexity of gesture recognition, the need for accurate and quick interpretation, and the variability in individual signing styles. A machine learning model that can process and translate sign language in real-time has the potential to bridge the communication gap in virtual settings, making online interactions more inclusive.

Relevance:
Solving this problem is highly meaningful as it contributes to making communication universally accessible, a necessity in our increasingly digital world. Accessibility in virtual meetings is vital for educational, professional, and social interactions, and sign language users often face barriers that limit their full participation. By implementing a machine learning-based solution, we can create a system that not only aids communication but also demonstrates the application of AI for social good. This project lies at the intersection of computer vision and accessibility, showing how advancements in technology can empower communities and foster inclusivity.

Method

Overview of the Approach:
To address the need for real-time sign language translation in virtual meetings, we leverage a robust American Sign Language dataset with 29 classes, covering letters a to z, deletion, nothing, and space. This dataset, with approximately 8,000 images per class, provides a solid foundation for training our machine learning model. Given the multi-class nature of this classification task, we conducted thorough research on various deep learning models suited for gesture recognition, ultimately selecting three models for comparative evaluation to identify the most accurate and efficient option.



Technical Details:

Model Architecture: We utilized a convolutional neural network (CNN)-based machine learning model because CNNs are particularly well-suited to image classification tasks such as sign language recognition. CNN based models excel at identifying spatial layers in an image, which is essential for understanding and distinguishing between different hand gestures in sign language. Their unique architecture allows them to capture and learn complex patterns such as edges, shapes, and textures from raw image data by applying a convolutional filter across the image.

Training Process: Our dataset consists of 29 classes—26 for the letters A through Z and 3 additional classes for “space,” “delete,” and “nothing.” As a result, the output layer of our model has 29 neurons, with softmax applied to produce class probabilities.

Since our training images are sized at 200x200, we were cautious about scaling them up to 224x224 (ResNet’s standard size) and instead opted to scale down to 128x128, the next best option for image transformations.

A key insight we applied was the use of grayscale images. Since the critical features for classification—such as curves, edges, and shapes—are independent of color, we converted images to grayscale. This approach also allows us to preprocess any new test images to grayscale before inputting them into the classifier, making our model more efficient without sacrificing essential details.

To adapt the ResNet-18 architecture for our needs, we modified the first layer to accept grayscale images and set the kernel size to 5. Additionally, we customized the last layer to accommodate our 29 classes. Other than these adjustments, the ResNet-18 architecture remained unchanged.

Leveraging ResNet-18 pretrained on ImageNet helped significantly; the pretrained filters are likely closer to those our model would learn for our dataset, making fine-tuning both effective and efficient.

Hardware and Resources: Since we lack access to high-performance servers with GPUs, we used Colab Pro to train our models. Colab’s GPU is beneficial but not particularly fast, so training time was a critical factor in selecting the model architecture. Colab Pro provided a balance between accessibility and computational power, allowing us to proceed with our model training within our resource limits.

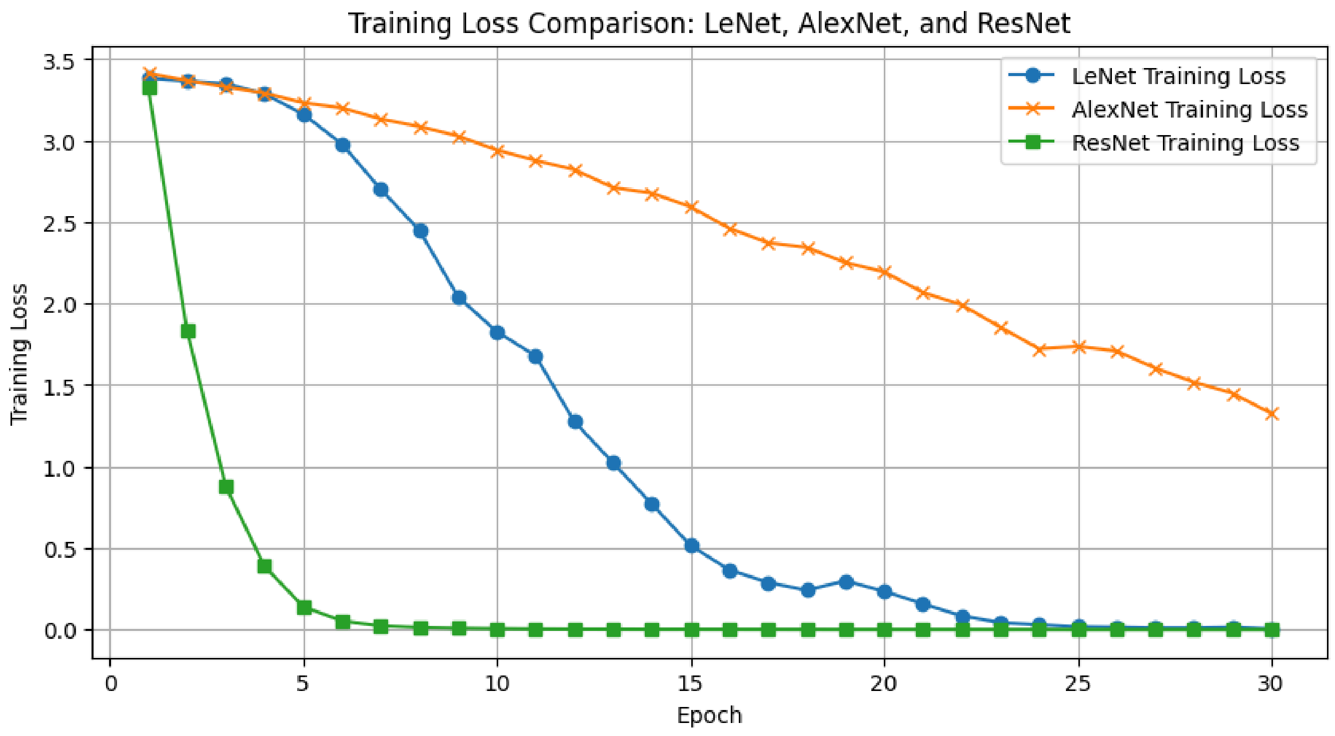
Preliminary Experiments

Due to limited resources (not enough compute), it was challenging to train and compare multiple models on a large dataset. So, we decided to create a mini dataset containing 10 images per class, totaling 290 images. We trained different models, from relatively simple to complex deep learning models, with this mini dataset and compared various performance metrics to determine the best model for our application.

Baseline Model: We used **LeNet** as the baseline model because it is relatively simple and serves as a good starting point for **CNN-based** neural networks.

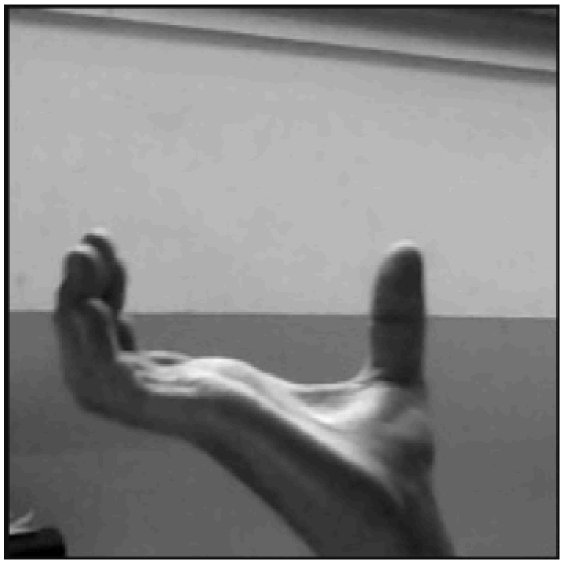
Advanced Models: We chose **AlexNet** and **ResNet-18** as advanced models for comparison, as both are well-known for their strong performance in multi-class image classification problems.

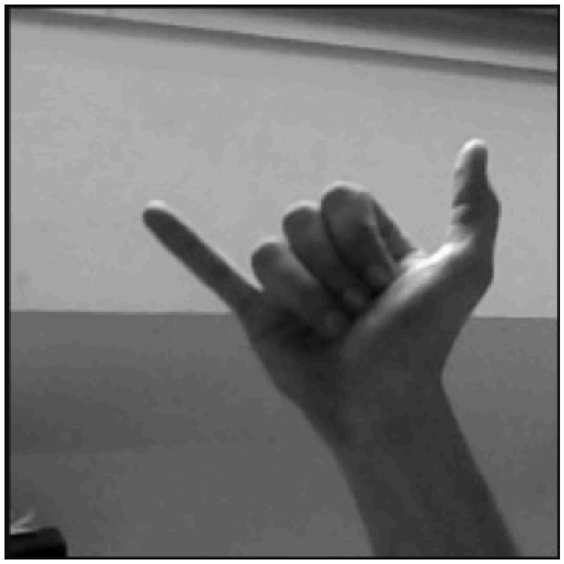
Training Loss:



Comparison of Key Metrics Across Different Models Trained on a Mini Dataset:

As seen in the above graph, all 3 models had converging loss which was good but we were quick to see that LeNet was not a strong contender because it could not capture the more *complex* features in our images. LeNet performed decently on distinctive letters but it was often confusing the letter *Y* and the space symbol.





This definitely seemed like an architecture issue where the model was not learning more complicated features and hence we decided to go with a more complicated model.

While accuracy is the most important factor, we also considered training time because of limited resources. Inference time was also crucial due to the real-time requirements of our application.

All things considered, we decided to use ResNet-18, as it shows high accuracy with reasonable training and inference times on the mini training and test datasets.

	LeNet	AlexNet	ResNet-18
Accuracy	13.45%	29.66%	67.59%
Training Time	51.98 (s)	53.62 (s)	67.92 (s)
Inference Time	1.49 (s)	1.48 (s)	1.73 (s)

This showed us that ResNet-18 was giving us pretty good values. This was trained on our MiniDataset which only had 10 images per class, showing us that ResNet-18 was learning our images pretty well.

Identifying Sign Language in larger images & Haar Cascade Classifier:

In future iterations of our experiments, we want to be able to process any image of anyone communicating with sign language. To be able to do this, we need to focus on the part of the image where the user's hand is. We have started off by using the Haar Cascade classifier with their hand detection model - *haarcascade_hand.xml*.

Depending on the accuracies, we will either stick with this model or train a more advanced object detection model like YOLO or maybe even a custom hand (or specifically sign language on hand) detection model.

Future Work

Planned Experiments: For our planned experiments, we aim to push the boundaries of our current model by exploring both video processing and natural language processing (NLP) integration. First, we plan to create a feature where a user can record a video of themselves performing sign language, which will then be converted into text. This experiment will allow us to handle continuous sequences of sign language gestures, moving beyond single images to full video streams. Additionally, we plan to incorporate an NLP component to enhance the text output from our sign language translation. For instance, if there is a small typo or grammatical error in the generated text, the NLP module can automatically correct it, improving readability and accuracy. Looking further ahead, we envision a real-time feature where the model processes sign language live and generates text on the screen instantly, enabling immediate communication.

Expected Challenges: A primary challenge we anticipate is computational power. While we have access to 100 compute units through Google Colab Premium with advanced GPU resources, we are concerned this may not be sufficient for our ambitions, particularly when running multiple models simultaneously. Our plan to incorporate object detection (possibly with a model like YOLO) alongside a sign language recognition model and an NLP correction model will likely require a significant amount of computational resources. Additionally, handling real-time processing for video input demands substantial GPU power and may push the limits of our available hardware. Balancing the computational load, optimizing our models, and efficiently managing resources will be critical to overcoming this challenge.

Refinement Goals: Our ultimate goal is to create a fully functioning, locally hosted model that enables quick and accurate sign language interpretation. Before final submission, we hope to refine our model to the point where it can handle near real-time processing, allowing users to see translated text from sign language gestures almost instantaneously. For our demo, we aim to set up a two-sided video interface where one person can communicate in sign language, while the other sees live text translations, facilitating a seamless, two-way conversation. This demonstration will showcase the potential for accessible and inclusive communication technology, allowing both hearing and non-hearing individuals to engage in real-time conversations.

References

1. He, K., Zhang, X., Ren, S., & Sun, J. (2016). "Deep Residual Learning for Image Recognition." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770-778.
    - This paper introduces ResNet, a deep learning model that uses residual learning frameworks to improve training in deep networks, forming the basis of our model's architecture.
2. Simonyan, K., & Zisserman, A. (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition." *International Conference on Learning Representations (ICLR)*.
    - A foundational paper on convolutional neural networks, which helped inform our understanding of image recognition techniques and architecture design.
3. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
    - This textbook covers the fundamentals of deep learning, including convolutional networks and fine-tuning methods, which have been instrumental in the development of our model.
4. Sutskever, I., Vinyals, O., & Le, Q. V. (2014). "Sequence to Sequence Learning with Neural Networks." *Advances in Neural Information Processing Systems (NeurIPS)*.
    - This paper discusses sequence learning, which has inspired our NLP component for handling text correction and grammatical adjustments in sign language translation.
5. Ronneberger, O., Fischer, P., & Brox, T. (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation." *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*.
    - Although focused on biomedical segmentation, U-Net’s architecture influenced our understanding of effective image segmentation techniques, which are relevant for detecting hands in sign language videos.