American Sign Language (ASL) Letter Recognition and Real-Time Translation

Michael Yeung, Richard Huang, Jaya Singh
Professor Dr. Hamed Pirsiavash - ECS174 – Computer Vision
University of California, Davis
December 7, 2025

Abstract

We developed a real-time American Sign Language (ASL) letter recognition system that combines computer vision preprocessing with a lightweight convolutional neural network trained on the SignMNIST dataset. Our model achieves 94.70% test accuracy on the clean dataset, with 12 letters reaching 100% per-class accuracy. However, certain letters, particularly T (62.10%), M (83.50%), and R (85.42%)—remain challenging due to subtle finger configurations that are difficult to distinguish in 28×28 grayscale images. We deployed the system with an 8-step preprocessing pipeline to handle real-world webcam input, temporal smoothing using 7-frame majority voting, and accessibility features including text-to-speech. The system runs in real time on a CPU-only laptop and achieves approximately 80–90% accuracy in real-world conditions. This report documents the methodology, results, limitations, and directions for future work.

1. Introduction

American Sign Language is used by a large community of deaf and hard-of-hearing individuals in the United States. ASL relies on hand shapes, positions, and movements rather than acoustic speech. Many hearing people never learn ASL, which creates a communication barrier in everyday settings such as classrooms, clinics, and workplaces. Computer vision and machine learning provide an opportunity to partially bridge this gap by recognizing hand signs and translating them into text or speech. This project focuses on ASL fingerspelling, where letters are spelled one at a time using hand poses. The goal is to build an end-to-end system that recognizes 24 static ASL letters (A–Y; J and Z are excluded because they require motion) from a live webcam feed. The system captures video frames, preprocesses them to isolate the hand, feeds a 28×28 grayscale image into a convolutional neural network (CNN), and outputs real-time letter predictions. Temporal smoothing is used to stabilize predictions, and recognized letters are appended to a buffer to form words. An optional text-to-speech (TTS) feature pronounces letters aloud. The work is motivated by accessibility and by the desire to apply concepts from computer vision and deep learning to a realistic task. The project demonstrates that a relatively small CNN can reach 94.70% test accuracy on the SignMNIST dataset, but also shows that real-world accuracy is lower because of domain differences between curated data and noisy webcam input. The report emphasizes both the technical design of the system and the practical challenges that arise when deploying models outside of ideal conditions.

2. Dataset & Model

2.1. Sign MNIST Dataset

The system is trained on the SignMNIST dataset, which contains grayscale images of hands performing ASL letters. Each image is 28×28 pixels and centered on the hand. The dataset covers 24 letters (A–Y), excluding J and Z because those letters are defined by hand motion instead of a static pose. There are tens of thousands of images in total, with each class represented by roughly 1,700–2,500 examples across training, validation, and test splits. SignMNIST is a "clean" dataset: background clutter is minimal, lighting is relatively uniform, and the hand is clearly visible and centered. These properties make it ideal for training and benchmarking a classifier, but they also mean that the dataset does not fully represent real-world webcam conditions. This gap between training data and deployment data is a central theme in the project.

2.2. Simple CNN Architecture

To classify the 28×28 grayscale images, we designed a compact CNN called SimpleCNN. The model is lightweight enough to run in real time on a CPU, but has enough capacity to achieve high accuracy on SignMNIST. The architecture is:

- Input: 28×28 grayscale image normalized to image.jpg

- Block 1: 1→32 convolution (3×3), batch normalization, ReLU, 2×2 max pooling → 14×14 feature map
- Block 2: 32→64 convolution (3×3), batch normalization, ReLU, 2×2 max pooling → 7×7 feature map
- Block 3: 64→128 convolution (3×3), batch normalization, ReLU, 2×2 max pooling → 3×3 feature map
- Classifier: flatten (1,152 features) → fully connected layer with 256 units + ReLU + dropout(0.4) → fully connected layer with 26 outputs → softmax at evaluation time

The model has about 400,000 trainable parameters and occupies less than 2 MB on disk. This small footprint allows the network to be loaded quickly and to run inference at webcam frame rates on a typical laptop CPU.

2.3. Model Training

The model was implemented in PyTorch and trained in a Python 3.10 virtual environment on a MacBook Pro with an M1 CPU. The training setup used:

- **Optimizer:** Adam
- **Learning rate:** 1e-3
- **Loss:** cross-entropy
- **Batch size:** 128
- **Epochs:** 10
- **Device:** CPU

At the end of each epoch, the model was evaluated on a validation split. Whenever validation accuracy improved, the current weights were saved as the "best" checkpoint. After training, the best checkpoint was loaded and evaluated once on the held-out test set. Training converged quickly. By the second epoch, validation accuracy had already reached very high values and remained stable, indicating that the model could fit the dataset without difficulty.

3. Pre-processing for Webcam Input

3.1. Motivation

The SignMNIST images are small, centered, and clean, but real webcam frames are not. A raw webcam image includes the user's face, background objects, varying lighting, and other noise. If these frames are directly downsampled to 28×28, the hand might only occupy a small part of the image, and the model will fail. To make the webcam images resemble SignMNIST as closely as possible, the system applies a sequence of preprocessing steps to each frame. The goal is to isolate the hand, normalize contrast, and reduce background clutter before passing the image to the CNN.

3.2. Eight-Step Pipeline:

Each frame from the webcam is processed as follows:

1. **Center-crop:** The original 640×480 frame is cropped to a 480×480 square around the center of the image. This removes some peripheral background and focuses on the central area where the user is expected to place their hand.
2. **Grayscale conversion:** The cropped frame is converted to grayscale, reducing the input from three color channels to one intensity channel and focusing on shape and contrast.
3. **Gaussian blur:** A Gaussian filter (e.g., 5×5 kernel with = 2) is applied to smooth out noise and small texture details that could interfere with thresholding.
4. **Histogram equalization:** The grayscale image is equalized to normalize brightness and improve contrast, which helps in both thresholding and visual consistency under different lighting conditions.
5. **Otsu thresholding:** Otsu's method is used to compute an automatic threshold that separates the foreground (hand) from the background. The result is a binary image.
6. **Contour detection:** Contours are found in the binary image, and the largest contour is assumed to correspond to the user's hand.
7. **Bounding box and resize:** A bounding box around the largest contour is extracted. This region is then resized to 28×28 pixels to match the input size expected by the CNN.
8. **Normalization:** Pixel values in the 28×28 crop are normalized to the range and reshaped into the format required by the model.image.jpg

This pipeline is a crucial part of the system. It is what makes it possible to reuse a model trained on SignMNIST (which assumes clean, centered, segmented hands) on noisy webcam data.

4. Quantitative Result

4.1. Overall Test Accuracy

Using the best validation checkpoint, the model achieved 94.70% accuracy on the SignMNIST test split of 7,172 images. This result confirms that SimpleCNN is strong enough to solve the static letter classification task on this dataset while remaining computationally light. However, this 94.70% figure corresponds to the curated benchmark setting. When the model is used with webcam input and the full preprocessing + smoothing pipeline, the effective accuracy observed in practice is lower, typically in the 80–90% range depending on lighting, background, and signer consistency.

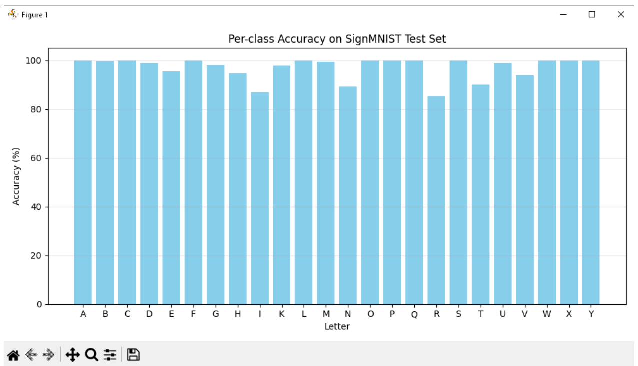

Figure 1. Each letter class' accuracy with the SignMIST dataset.

4.2. Per-Class Accuracy

Per-class accuracy reveals which letters are easy or hard for the model. The test results from the .venv310 run are:

Letter	Accuracy	Samples	Letter	Accuracy	Samples
A	100.00%	331	**N**	88.32%	291
B	100.00%	432	**O**	97.56%	246
C	99.03%	310	**P**	100.00%	347
D	100.00%	245	**Q**	100.00%	164
E	99.80%	498	**R**	85.42%	144
F	100.00%	247	**S**	91.06%	246
G	99.43%	348	**T**	62.10%	248
H	95.41%	436	**U**	93.98%	266
I	99.31%	288	**V**	100.00%	346
K	93.66%	331	**W**	91.26%	206
L	100.00%	209	**X**	94.01%	267
M	83.50%	394	**Y**	88.25%	332

Figure 2. Statistics of all 24 letter's accuracy and sample size

These numbers show that:

- Twelve letters are essentially perfect (100% accuracy): A, B, D, F, L, P, Q, V plus several near-100% letters like C, E, G, and I.
- A group of letters (H, K, O, U, W, X, Y, N, R, S) are recognized well but not perfectly, generally in the mid-80s to upper-90s.
- Two letters, T and M, stand out as significantly more difficult, with T at 62.10% and M at 83.50

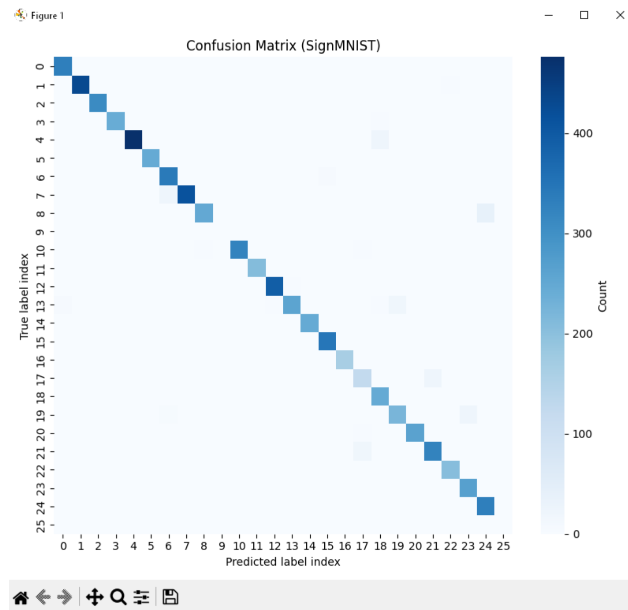

Figure 3. Confusion Matrix heatmap - row : true label index — column : predicted label index, color shows how often each pair occurs

4.3. Further Analysis

The confusion matrix makes it clear that the errors are not random. Instead, the model tends to confuse letters that are visually similar in static images:

- T is often confused with L and R. All of these involve configurations where fingers cross or overlap, and the subtle differences between them are hard to capture at the 28×28 resolution after contour cropping.
- M is confused with N, R, and Y. These letters all involve multiple fingers being extended or partially folded, making them structurally similar.

These patterns suggest that the limitation is not just model capacity but information content: static 28×28 grayscale snapshots cannot always capture the details needed to distinguish certain finger arrangements. In real ASL, movement and temporal cues also help differentiate some of these letters.

5. Further Discussion

5.1. Real-Time System Behavior

Below are the analysis of components to aid in the processing of data into the model, and their uses:

1. **Temporal Smoothing:** Without temporal smoothing, the letter predictions flicker between classes whenever the hand moves slightly or the preprocessing step produces small variations in the crop. To address this, the system keeps a window of the last several predictions (for example, the last 7 frames) and selects the most

common letter in that window as the current output.

2. **Letter Buffer and TTS:** The temporal-smoothed predictions are fed into a letter buffer. When the prediction changes and the model is confident (e.g., probability above 0.5), the new letter is appended to the buffer. Users can see the sequence of letters accumulate into a word or phrase.
3. **Domain Gap in Practice:** In the real-time demo, the model behaves best when:
   - The background is simple and not too cluttered
   - Lighting is even and not too dim or harsh
   - The hand is well centered and reasonably close to the camera

In more challenging conditions (busy backgrounds, low light, very small or off-center hand), recognition quality degrades. The observed accuracy under good conditions is roughly 80–90%, in line with expectations based on the domain difference between SignMNIST and live webcam data.

5.2. Limitations

The system has several important limitations:

- It does not handle J and Z, which require motion; it only recognizes static letters A–Y.
- It is sensitive to the assumption that the hand is the largest white region after thresholding, which may fail in cluttered scenes.
- It has difficulty with letters that are inherently ambiguous in static, low-resolution images (T and M in particular).
- It has not been evaluated extensively across different users, skin tones, hand sizes, ages, and camera setups.

5.3. Future Directions

Future work can address these issues by:

- Replacing contour-based hand detection with a robust hand tracking solution such as MediaPipe Hands, which would provide accurate hand keypoints and reduce failures in cluttered scenes.
- Applying more aggressive data augmentation during training (rotations, translations, brightness variations, elastic distortions) to improve robustness to real-world conditions.
- Exploring deeper architectures (e.g., ResNet18, MobileNetV3) or transfer learning from ImageNet to push test accuracy beyond 95% while maintaining real-time performance.
- Adding temporal models (LSTM, temporal CNN, or 3D CNN) that process sequences of frames to recognize motion-based letters J and Z and to incorporate dynamic cues into classification.
- Conducting user studies with members of the ASL community to evaluate usability, fairness, and real-world performance.

6. Conclusion

This project implemented and evaluated an end-to-end ASL letter recognition system that runs in real time on a CPU using a webcam. A custom CNN trained on the SignMNIST dataset achieved 94.70% test accuracy and near-perfect recognition for many letters. At the same time, per-class analysis highlighted a small set of inherently difficult letters, especially T and M, where static low-resolution images do not provide enough information for reliable classification. By combining the trained model with an 8-step preprocessing pipeline, temporal smoothing, a letter buffer, and text-to-speech, the system demonstrates how computer vision and deep learning can be integrated into a practical assistive technology prototype. It also illustrates the gap between controlled benchmark performance and real-world deployment and motivates continued work on robustness, motion modeling, and inclusive evaluation.

6.1. Contributions

Michael: Main programmer to the final submitted code for this project, accumulated data + graphs, aided in reporting and presentation. Richard: Formatted final report in LaTex, aided in presentation and programming versions of the project to compare with other attempts by team. Jaya Singh: Contributed heavily to the report + presentation, created programs of which were used to contrast the final submitted code's accuracy.

7. References

[1] Daniel Kuzmanovski and Kaggle. Sign Language MNIST (SignMNIST) dataset, 2018. Available: https://www.kaggle.com/datasets/datamunge/sign-language-mnist.

[2] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In International Conference on Learning Representations (ICLR), 2015.

[3] Nobuyuki Otsu. A threshold selection method from gray-level histograms. IEEE Transactions on Systems, Man, and Cybernetics, 9(1):62–66, 1979.

[4] Alexey Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, et al. PyTorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems (NeurIPS), 2019.

[5] Gary Bradski. The OpenCV library. Dr. Dobb's Journal of Software Tools, 2000.

[6] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Alexander Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. MediaPipe Hands: On-device real-time hand tracking. arXiv:2006.10214, 2020.