Real-Time ASL Alphabet Classification with ResNet18 and Hand-Tracked ROI

Max Hansen
Department of Electrical and Computer Engineering
University of Minnesota
Minneapolis, MN, USA
hans8439@umn.edu

***Abstract*—I build an American Sign Language (ASL) alphabet classifier using a ResNet18 backbone and deploy it in a real-time webcam demo. The system performs image classification on a region-of-interest (ROI) extracted from each camera frame. To improve robustness under hand motion, I integrate MediaPipe Tasks hand landmark detection to dynamically crop a padded hand bounding box, falling back to a centered ROI when detection fails. I train the model for 8 epochs on clean training images using Minnesota Supercomputing Institute (MSI) GPU nodes and evaluate performance on held-out data. The demo includes moving-average probability smoothing for stable live predictions. I discuss limitations related to background cleanliness, hand pose angle, and ROI shifts.**

***Index Terms*—ASL, accessibility, image classification, ResNet18, real-time inference, hand landmarks, ROI tracking**

I. Introduction and Motivation

American Sign Language recognition is a practical computer vision task with applications in accessibility and human–computer interaction. I chose this project with two motivations: (1) accessibility - ASL tools can help bridge communication gaps and support inclusive interaction; and (2) learning opportunity - building an end-to-end recognition system provides hands-on experience with modern deep learning workflows, deployment constraints, and real-time perception.

This project focuses on real-time classification of ASL letters signed by the user. My goal was to demonstrate how far a relatively simple ResNet-based classifier [1] can go when it is paired with a practical, end-to-end webcam pipeline. ResNet (short for *residual network*) is a widely used architecture for image classification because it can be trained effectively at depth and tends to learn strong visual features. It is called "residual" because the network is built from residual blocks that learn a *residual function* rather than a direct mapping: instead of learning $H(x)$ directly, each block learns $F(x)$ and combines it with the original input through an identity skip connection (conceptually $y = F(x) + x$). This block structure can be seen in figure 1. These shortcuts improve optimization by enabling smoother gradient flow and by allowing layers to focus on learning incremental refinements of the representation rather than re-learning the entire transformation.

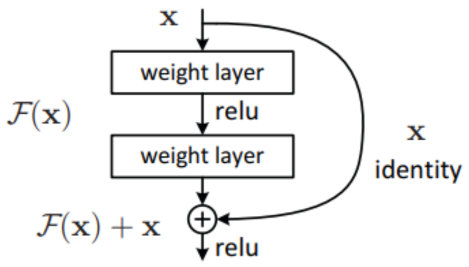

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Fig. 1: Residual block in a ResNet: the input x is added to the learned residual mapping $\mathcal{F}(x)$ via an identity skip connection, producing $\mathcal{F}(x) + x$ and improving gradient flow for training deeper networks.

To make the model usable in an interactive setting, I built a webcam application that runs end-to-end inference in real time and overlays the predicted letter (and confidence) on the live video stream. A central challenge in webcam demos is that naive cropping is brittle: small hand translations, changes in distance to the camera, or slight shifts in framing can move relevant pixels out of the classifier input and lead to unstable predictions. To reduce this sensitivity, I incorporated hand landmark detection to estimate the hand location each frame and used the detected hand region as a dynamic region-of-interest (ROI). By updating the ROI frame-by-frame, I produce more consistent, centered inputs to the classifier, which improves robustness and makes the live demo significantly more reliable.

II. Dataset

A. Source

I use the *ASL Alphabet* image dataset published on Kaggle [2]. The dataset contains labeled images of ASL hand signs captured under relatively controlled conditions (clean backgrounds and consistent framing). For my final training runs, I focus on a "clean" subset of training images to reduce label noise and domain mismatch during deployment.

B. Classes and Splits

The classifier predicts C classes corresponding to ASL symbols (e.g., A–Z plus special tokens such as `space`, `del`, and `nothing`, matching the class list used in `src/labels.py`). I create train/validation/test splits with approximately stratified class proportions. I report accuracy on the held-out test split.

C. Preprocessing

For validation and deployment, each input image (or cropped ROI) is resized to 224×224 and normalized using ImageNet mean and standard deviation:

$$\mu = [0.485, 0.456, 0.406], \quad \sigma = [0.229, 0.224, 0.225].$$

This matches standard ResNet preprocessing and is used consistently across evaluation and the webcam demo.

III. Method

A. ResNet18 Classifier Architecture

I use ResNet18 [1], a residual convolutional neural network composed of an initial convolution + max-pooling stem followed by four stages of residual blocks. Each residual block learns a residual mapping added to the block input via a skip connection:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}) + \mathbf{x}, \tag{1}$$

where $\mathcal{F}(\cdot)$ is a small stack of convolutions, batch normalization, and ReLU activations. The skip connections improve optimization stability and enable deeper networks to train effectively.

ResNet18 uses BasicBlocks and a stage configuration of $\{2,2,2,2\}$ blocks, with channel widths increasing across stages. The network ends with global average pooling and a final fully-connected layer producing logits $\mathbf{z} \in \mathbb{R}^C$.

B. Training Objective

Given logits $\mathbf{z}$, I compute softmax probabilities

$$p_k = \frac{e^{z_k}}{\sum_{j=1}^{C} e^{z_j}}, \tag{2}$$

and minimize cross-entropy loss for the ground-truth class y:

$$\mathcal{L} = -\log p_y. \tag{3}$$

IV. Training and Optimization Details

A. Compute Environment

I trained the model on Minnesota Supercomputing Institute (MSI) GPU nodes. Using MSI allowed faster iteration and experimentation than CPU-only training.

B. Training Setup

I trained for 8 epochs using a "clean" training subset (images with relatively consistent framing and background). This choice improved stability and reduced confusing cases but also increased the gap between training distribution and real-world webcam conditions.

C. Hyperparameters

My final run used:

- Epochs: 8
- Optimizer: `Adam`
- Learning rate: `3e-4`
- Batch size: `128`
- Weight decay: `3e-4`
- Input size: 224

(These settings match the final demo checkpoint used for deployment.)

V. Real-Time Webcam Demo

A. Overview

The webcam application repeatedly (i) captures a frame, (ii) selects an ROI, (iii) preprocesses, (iv) runs inference, and (v) renders overlays. The interface displays the predicted label, confidence, top-k classes, and FPS.

B. Hand Landmarking for ROI Selection

To reduce sensitivity to hand position shifts, I integrate MediaPipe [3] Tasks hand landmarking to localize the hand. The hand landmarker returns 21 landmarks in normalized image coordinates $(x_i, y_i) \in [0,1]^2$. I compute a bounding box around the landmarks and apply padding to include the full hand and some context:

$$x_{\min} = \min_i x_i, \quad x_{\max} = \max_i x_i, \quad y_{\min} = \min_i y_i, \quad y_{\max} = \max_i y_i. \tag{4}$$

The padded ROI is cropped from the frame and fed to the classifier. If no hand is detected, the system falls back to a centered square ROI.

C. Temporal Smoothing

To reduce flicker, I smooth the predicted probabilities with a moving average over the last N frames:

$$\bar{\mathbf{p}} = \frac{1}{N} \sum_{t=1}^{N} \mathbf{p}^{(t)}. \tag{5}$$

I display the top-1 label if its confidence exceeds a threshold; otherwise the demo displays "?".

D. Run Command

```
python -m src.webcam_demo --ckpt checkpoints/best.pt
     ^
  --hand-landmarks --mp-model models/hand_landmarker
      .task ^
  --draw-roi --mirror
```

VI. Results

A. Webcam Demo Results

Under controlled conditions similar to the training set: clean background, stable framing, and a clearly posed hand within the ROI, the classifier produces confident and stable predictions. The landmark-driven ROI improves the user experience by tracking the hand, allowing more natural movement compared to a fixed center crop. Temporal smoothing further reduces prediction flicker and makes the live output easier to interpret. I evaluated my trained classifier in a live webcam demo that performs hand detection/ROI cropping and then predicts an ASL letter from the cropped hand image. Qualitatively, I observed stable real-time predictions at roughly 30 FPS (as indicated by the on-screen overlay) and frequent high-confidence correct classifications. Figure 2 shows twelve representative *correct* predictions spanning a range of hand shapes (e.g., `K, B, C, L, Y, O, U, W, M, N, E, R`). In many of these examples, the correct letter is ranked first with a large margin over the next-best candidates, suggesting the network learned useful geometric cues in the hand pose (finger count/spacing, thumb placement, and overall silhouette) when the input resembles my training distribution. To sanity-check generalization outside of the training split, I evaluated the final checkpoint on the Kaggle ASL Alphabet "test" directory, which contains one representative image per class (26 letters plus `del`, `nothing`, and `space`; 28 images total). Using the same preprocessing pipeline as training/inference, the model achieved **28/28 correct** for **Top-1 accuracy = 1.00** and **Top-3 accuracy = 1.00**. In addition to the perfect Top-1 predictions, the Top-3 lists were consistently reasonable (e.g., `C` competing with visually similar shapes such as `O`, and `M`/`N` appearing as close alternatives), which suggests the network's learned decision boundaries align with intuitive pose similarities.

a) Interpretation and caveats.: While this result is encouraging, I treat it primarily as a *pipeline and label-mapping validation* rather than a definitive measure of performance. This Kaggle "test" folder is extremely small (one image per class) and is visually similar in style to the training imagery, so it does not adequately capture variation in subjects, viewpoints, lighting, backgrounds, or camera quality. As a result, a perfect score here should not be interpreted as "solving" the task; it mainly confirms that the trained model, preprocessing, and class ordering are consistent and that the classifier can generalize to at least one unseen exemplar per class drawn from the same overall distribution. More realistic testing would require a larger held-out test split and/or an explicit distribution-shift evaluation (e.g., webcam-like images with varied angles/lighting/backgrounds) to quantify the sensitivity observed in the live demo.

B. Missed Results (Failure Cases)

The main failure cases arise from distribution shift between training images and live webcam input:

- **Background clutter:** Busy backgrounds introduce textures and edges that the model may partially latch onto, reducing confidence or causing incorrect predictions.
- **Hand movement / ROI shift:** Rapid motion creates blur and can briefly shift the ROI, producing unstable logits even with smoothing.
- **Hand angle and pose:** Many letters are view-dependent; angling the hand away from the camera (or partially occluding fingers) can change key cues and lead to misclassification.
- **Similar gestures:** Some letters are visually similar in static images and can be confused without context.

VII. Limitations

Although the webcam demo is functional and responsive, performance depends on user behavior and environment. The system works best when the user presents a single hand clearly inside the ROI, holds the pose steadily, and uses a clean background. Generalization to varied lighting, backgrounds, and unconstrained hand motion remains a key limitation of training on a relatively controlled dataset.

VIII. Future Work

A natural extension is to make the system more interactive: build an application that guides users through learning ASL by prompting a target letter, providing real-time feedback (confidence, corrections), and tracking progress over time. This could be packaged as a desktop/mobile app with a user interface and lesson structure.

Beyond single letters, recognizing words and continuous signing would require temporal modeling and richer datasets. Possible directions include:

- **Sequence modeling:** extend from per-frame classification to short video clips using 3D CNNs or transformer-based video encoders.
- **Expanded vocabulary:** incorporate multi-letter sequences and common ASL words (challenging due to dynamic motion).
- **Stronger robustness:** add augmentations and domain randomization for lighting/background, or fine-tune with webcam-collected data.
- **Improved ROI policy:** incorporate hand orientation or handedness, multi-hand scenarios, and confidence-based gating (only update prediction when detection is stable).

IX. Conclusion

I presented a ResNet18-based ASL alphabet classifier trained on MSI GPU nodes and deployed in a real-time webcam demo. Dynamic ROI selection using hand landmarks and temporal probability smoothing improves stability and usability in live inference. The system performs best in controlled conditions and highlights the broader challenge of domain shift between curated datasets and unconstrained real-world input.

Appendix

This appendix provides additional qualitative examples from the real-time webcam demo. Each tile is labeled with the intended letter.

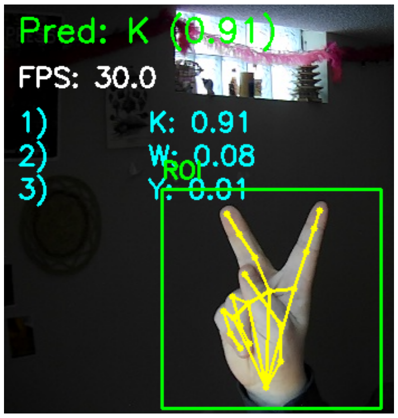
A. Correct Predictions (12 Examples)

References

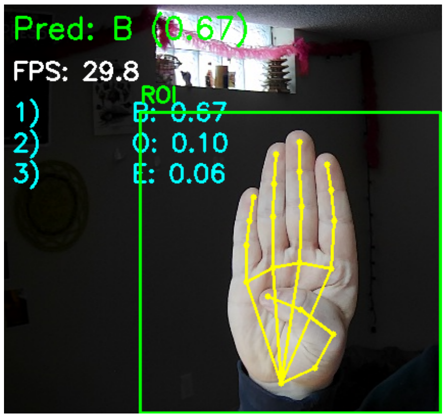
[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[2] Kaggle user, "Asl alphabet," Kaggle dataset, accessed 2025. [Online]. Available: https://www.kaggle.com/datasets/grassknoted/asl-alphabet

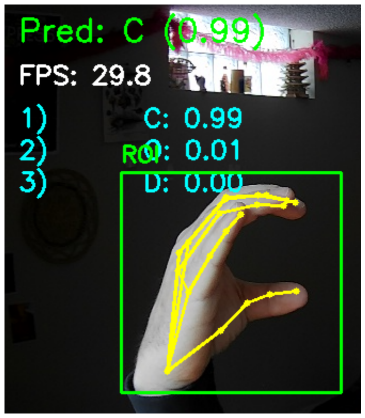
[3] Google AI Edge, "Hand landmarks detection guide," MediaPipe Tasks Vision documentation (Hand Landmarker), accessed 2025. [Online]. Available: https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker



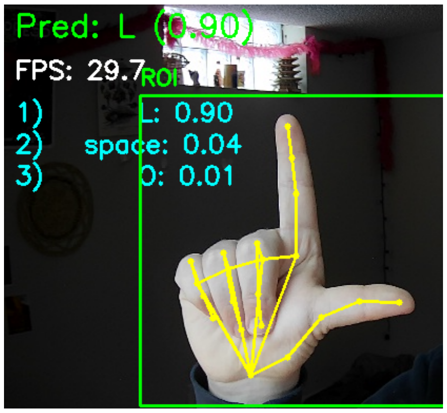
(a) Intended: K



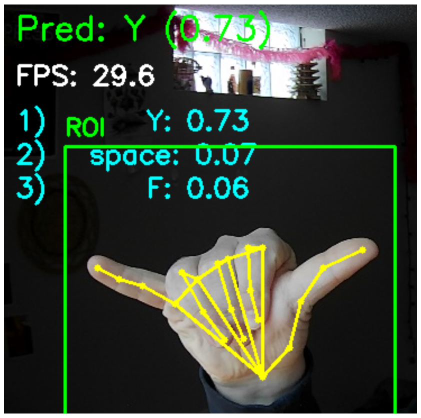
(b) Intended: B



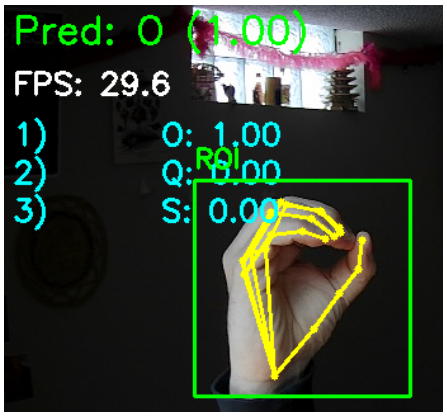
(c) Intended: C



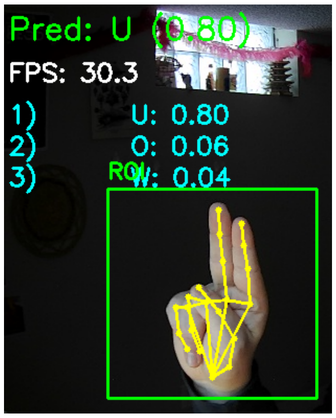
(d) Intended: L



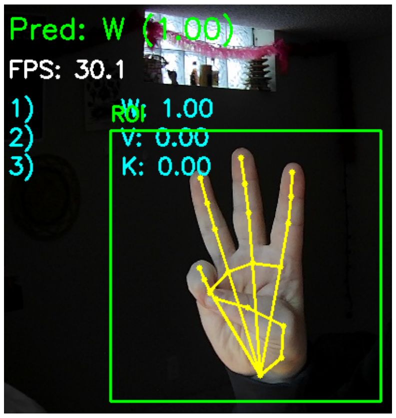
(e) Intended: Y



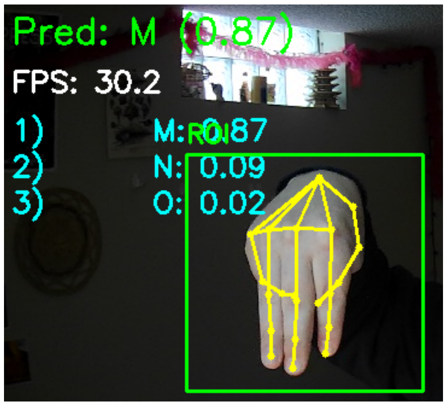
(f) Intended: O



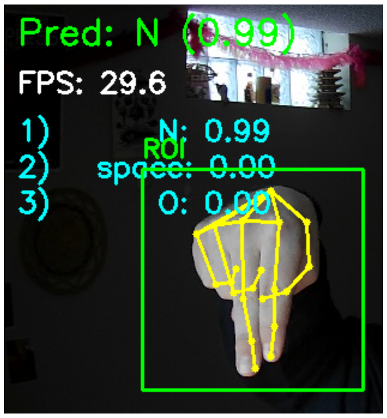
(g) Intended: U



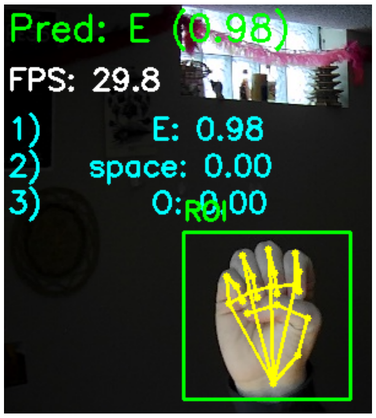
(h) Intended: W



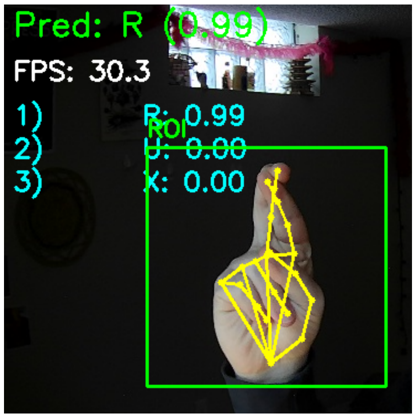
(i) Intended: M



(j) Intended: N



(k) Intended: E



(l) Intended: R

Fig. 2: Twelve correct predictions from the webcam demo.