CST-440: Gesture Recognition System

Jack Utzerath, Hunter Jenkins, Ricardo Escarcega

College of Technology, Grand Canyon University

CST-440: Analytics for Dynamic Social Networks Lecture & Lab

Kelly C

1/26/25

Code:

https://github.com/utzerath/Sign-Language-Reader

Report

Building a robust hand gesture recognition system for American Sign Language (ASL) requires careful attention to both the data and the neural network architecture. The overarching goal is to classify static hand signs, such as digits (0–9) and letters (a–z), by training a convolutional neural network (CNN) that can process images of the hand in real time. In this work, I constructed a CNN from scratch, carefully designed data augmentation strategies, implemented real-time detection via a webcam, and developed a script to guide users in capturing new sign images. By combining these elements, the project successfully recognizes signs in real-world conditions with a validation accuracy of approximately 82% after ten epochs of training.

The first critical step was assembling a dataset that captures the variety of signs. I created a directory structure called asl_dataset, wherein each subfolder corresponds to a particular sign (e.g., asl_dataset/a/, asl_dataset/b/, and so forth). Each subfolder stores JPEG images of that sign, often collected from multiple sessions and individuals. This structure allows automated labeling: each folder name is treated as a class label. However, one often finds that publicly available or initial datasets may not fully reflect real-world conditions—different lighting, backgrounds, or personal hand shapes. To address this, I built a pre-train picture-taking script. This script loops through each sign label, displays a reference image if one exists, shows a three-second countdown on the webcam feed, and then captures multiple images of the user's hand forming that sign. It relies on MediaPipe to locate the user's hand, computing a bounding box around it,

and automatically crops the image to isolate the hand region. The script then saves these cropped images back into the appropriate subfolder (e.g., asl_dataset/a/). Through this process, users can quickly personalize and expand their datasets, thereby increasing model robustness.

Once I had a suitably large and diverse dataset, I turned to preprocessing. Because images came in different resolutions and shapes, I standardized them to 224×224 pixels. Additionally, I normalized the pixel values so that the CNN could more easily learn meaningful features rather than being confounded by arbitrary numeric ranges. To further enhance generalization, I applied random data augmentation. In particular, I used random flips, rotations, zoom, and contrast changes. These random transformations allow the CNN to learn invariances and better handle real-world variability, thereby reducing overfitting to specific backgrounds or angles.

Designing and training the CNN was the next phase. Conceptually, my network is a multi-layered architecture that extracts hierarchical features from images. Early layers learn edges and simple textures, while deeper layers capture more complex patterns like the shape of fingers or the curvature of a palm. Specifically, the CNN includes convolution blocks followed by batch normalization, which helps stabilize and accelerate training. After progressively downsampling spatial dimensions, I employ global average pooling to reduce the feature map to a compact vector, which is then passed through dropout to prevent the model from memorizing training samples too precisely. The final dense layer outputs a probability distribution over all the classes—digits and letters—using a softmax activation function. Altogether, the network has around 2.3 million parameters, reflecting its capacity to recognize subtle differences among the 36 classes.

During training, I loaded images from asl_dataset/ with an 80/20 train-validation split, shuffled them, and batched them for efficient GPU processing. Over ten epochs, the network's training accuracy steadily rose to around 84%, while validation accuracy approached 82%. The validation loss reached about 0.56, indicating that the model was learning robust features without overfitting too severely. To monitor progress, I plotted training and validation accuracy and loss curves. Early in training, validation accuracy lagged behind training accuracy, but data augmentation and dropout helped close the gap to a manageable level. After ten epochs, the model consistently classified most signs correctly, demonstrating that the architecture and preprocessing pipeline worked well in tandem.

For real-time testing, I employed a MediaPipe-based hand detector that runs on each webcam frame. It identifies key landmarks in the user's hand and computes the bounding box. This bounding box is expanded slightly (by a margin of 20 pixels) to ensure the entire hand is captured. The cropped image is then resized to 224×224, normalized according to the same scheme used in training, and passed into the CNN for classification. If the top prediction's confidence exceeds 0.8, the system declares a detected sign (e.g., "Detected: a (0.92)"). Otherwise, it displays a "Low confidence" message. To further enhance usability, I integrated a grammar/spell checker that runs after each sign is appended to a text buffer. The buffer thus gradually accumulates letters into words or sentences, with on-the-fly corrections for common typos.

The system's final pipeline is straightforward to use. By running a command like python sign_lang_transfer.py train, one triggers the entire training routine on the local dataset. The script

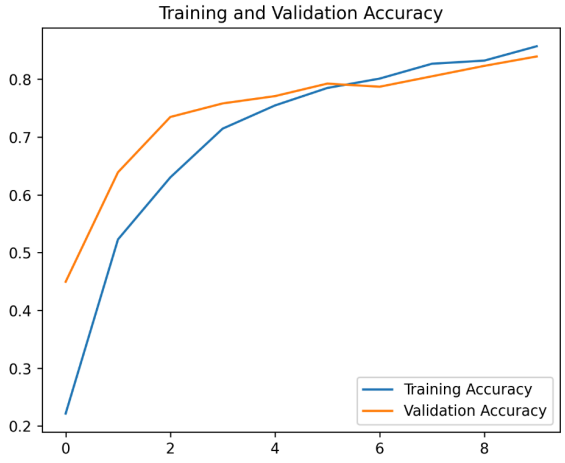
reports progress at each epoch, plots accuracy/loss curves, and saves the trained model in .keras format. Subsequently, running python sign_lang_transfer.py test opens the webcam feed, processes each frame with MediaPipe, crops the hand region, classifies it with the CNN, and displays results in real time. Pressing q terminates the session. If the user finds that the model struggles with certain backgrounds or personal hand geometry, they can capture more images with the pre-train script. By simply placing their hand in front of the camera for each letter, the system adds more examples to the dataset, improving the model's performance after retraining.

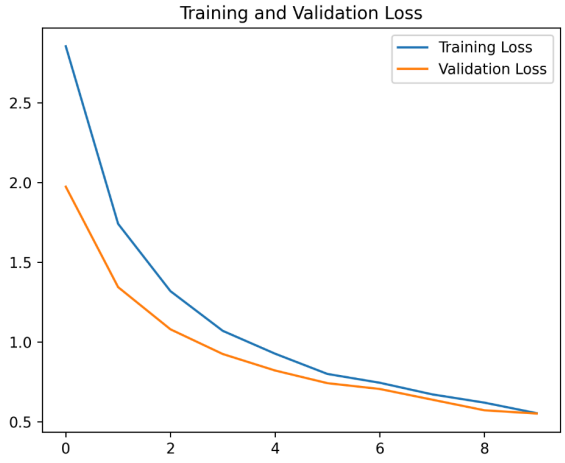
In conclusion, the project successfully demonstrates how to build a CNN for static ASL sign classification and integrate it with a real-time detection pipeline. The multi-step process—collecting data, preprocessing, training, validation, and final testing—ensures that each component works in harmony. The user-friendly script for gathering additional images further bridges the gap between controlled datasets and real-world conditions. With a validation accuracy of around 82%, the model already shows solid potential for practical applications, while the pipeline's modular design allows for straightforward enhancements, such as refining data augmentation, increasing epochs, or gathering more user-specific images. Ultimately, this approach highlights the feasibility of robust ASL recognition using a custom-built convolutional architecture, showcasing how machine learning techniques can advance inclusivity and communication in everyday settings.

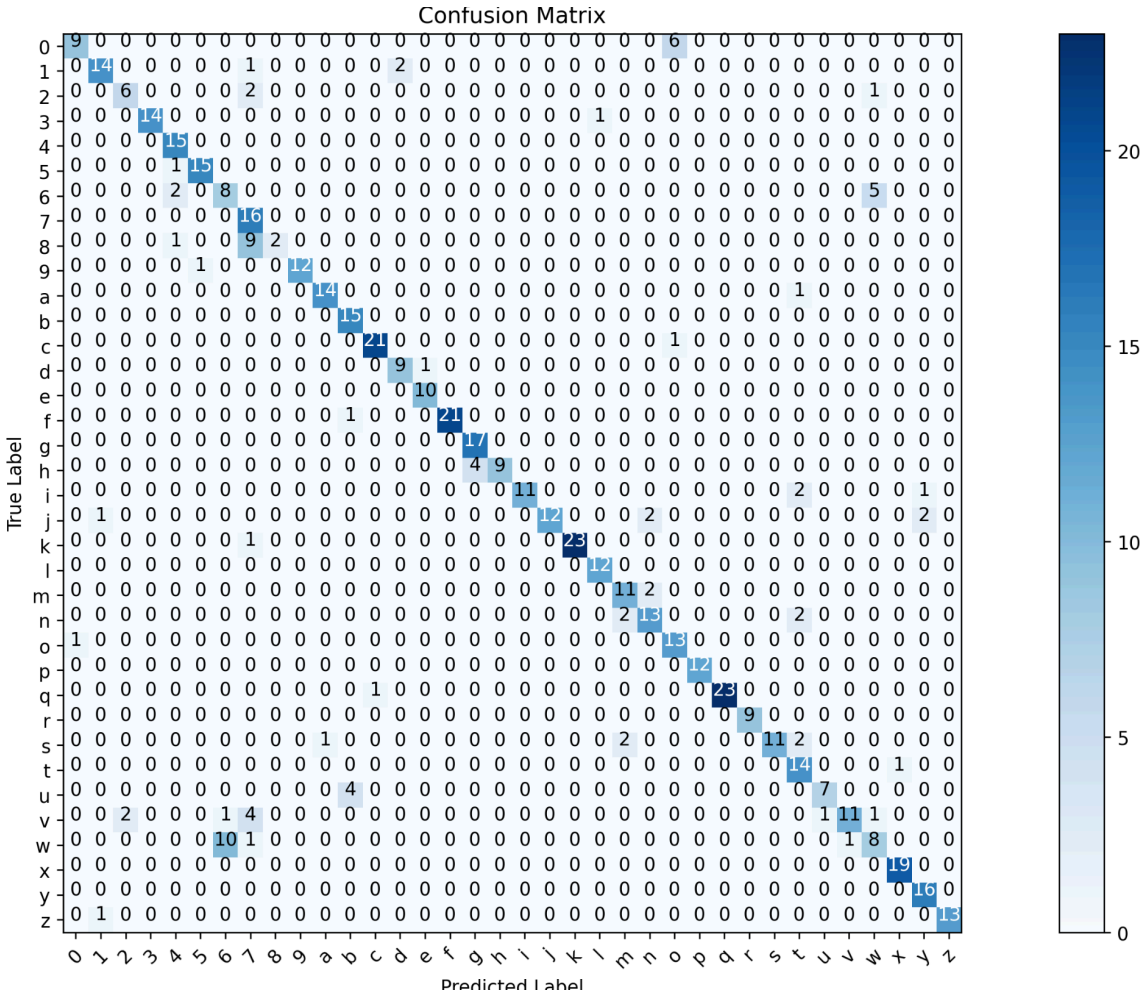
Output

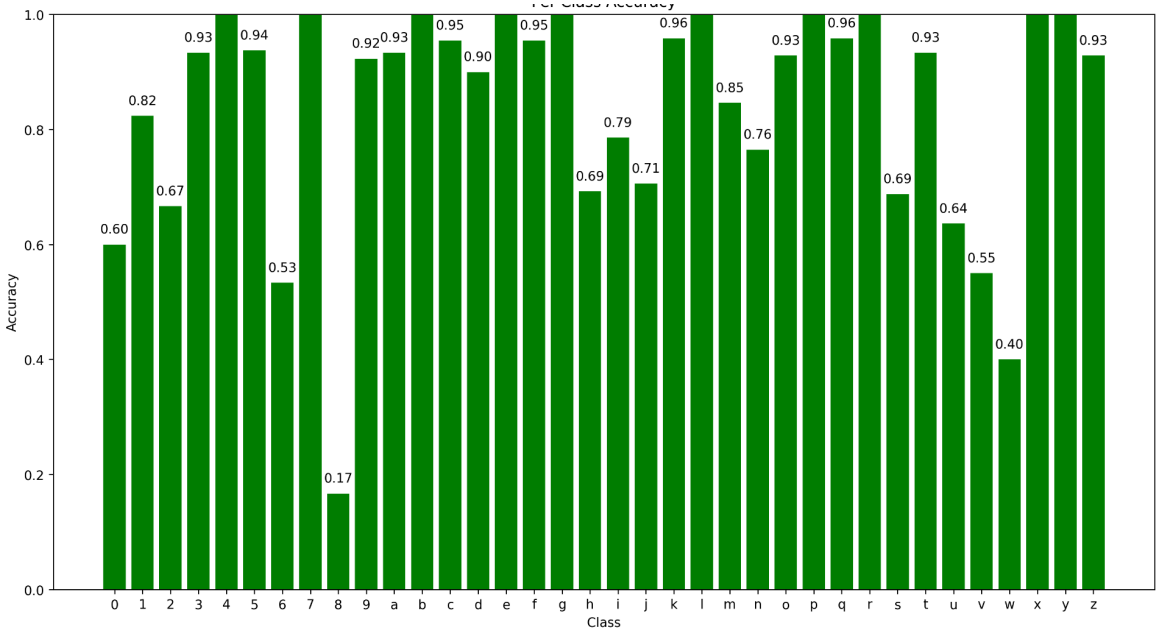
Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0
sequential (Sequential)	(None, 224, 224, 3)	0
true_divide (TrueDivide)	(None, 224, 224, 3)	0
subtract (Subtract)	(None, 224, 224, 3)	0
mobilenetv2_1.00_224 (Functional)	(None, 7, 7, 1280)	2,257,984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dropout (Dropout)	(None, 1280)	0
dense (Dense)	(None, 36)	46,116

```
Epoch 1/10
70/70 ━━━━━━━━━━━━━━━━━━━━ 43s 558ms/step - accuracy: 0.1215 - loss: 3.3531 - val_accuracy: 0.3682 - val_loss: 2.0329
Epoch 2/10
70/70 ━━━━━━━━━━━━━━━━━━━━ 35s 498ms/step - accuracy: 0.4765 - loss: 1.8840 - val_accuracy: 0.6480 - val_loss: 1.3727
Epoch 3/10
70/70 ━━━━━━━━━━━━━━━━━━━━ 36s 515ms/step - accuracy: 0.6122 - loss: 1.4089 - val_accuracy: 0.7238 - val_loss: 1.0747
Epoch 4/10
70/70 ━━━━━━━━━━━━━━━━━━━━ 37s 524ms/step - accuracy: 0.6888 - loss: 1.1248 - val_accuracy: 0.7274 - val_loss: 0.9307
Epoch 5/10
70/70 ━━━━━━━━━━━━━━━━━━━━ 35s 502ms/step - accuracy: 0.7551 - loss: 0.9461 - val_accuracy: 0.7545 - val_loss: 0.8187
Epoch 6/10
70/70 ━━━━━━━━━━━━━━━━━━━━ 34s 489ms/step - accuracy: 0.7782 - loss: 0.8434 - val_accuracy: 0.7780 - val_loss: 0.7454
Epoch 7/10
70/70 ━━━━━━━━━━━━━━━━━━━━ 33s 468ms/step - accuracy: 0.7864 - loss: 0.7796 - val_accuracy: 0.8051 - val_loss: 0.6673
Epoch 8/10
70/70 ━━━━━━━━━━━━━━━━━━━━ 32s 464ms/step - accuracy: 0.8097 - loss: 0.6891 - val_accuracy: 0.8177 - val_loss: 0.5904
Epoch 9/10
70/70 ━━━━━━━━━━━━━━━━━━━━ 32s 463ms/step - accuracy: 0.8309 - loss: 0.6516 - val_accuracy: 0.8032 - val_loss: 0.6057
Epoch 10/10
70/70 ━━━━━━━━━━━━━━━━━━━━ 36s 516ms/step - accuracy: 0.8500 - loss: 0.5804 - val_accuracy: 0.8231 - val_loss: 0.5559
18/18 ━━━━━━━━━━━━━━━━━━━━ 7s 375ms/step - accuracy: 0.8446 - loss: 0.5478
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy


Training and Validation Loss
Training Loss
Validation Loss


Confusion Matrix
True Label
Predicted Label


Accuracy
Class
0.60
0.82
0.67
0.93
0.94
0.53
0.17
0.92
0.93
0.95
0.90
0.95
0.69
0.79
0.71
0.96
0.85
0.76
0.93
0.96
0.69
0.93
0.64
0.55
0.40
0.93