Internship Report

Sign Language Detection System

By Tarrush Saxena

Task 5

Completed: January 2025

1. Introduction

This report documents my work on developing a Sign Language Detection System during my internship. The project aimed to create a real-time application that can recognize American Sign Language (ASL) alphabet gestures using computer vision and deep learning techniques. Sign language serves as a primary mode of communication for the deaf and hard-of-hearing community, and building tools that can interpret these gestures helps bridge communication gaps between sign language users and those unfamiliar with it.

The system I developed uses a webcam to capture hand gestures and processes them through a trained neural network to predict which letter is being signed. The final product is a desktop application with a graphical interface that displays predictions in real-time.

2. Background

Sign language recognition has been an active research area in computer vision for several years. Traditional approaches relied on sensor-based gloves or specialized hardware, but recent advances in deep learning have made camera-based recognition practical. The ASL alphabet consists of 26 letter signs plus additional symbols like space, delete, and nothing, making it a 29-class classification problem.

Before starting this project, I researched existing solutions and datasets. The ASL Alphabet dataset from Kaggle provided the training data needed, containing thousands of labeled images for each sign. I also studied transfer learning approaches, which allow using pre-trained models as a starting point rather than training from scratch.

3. Learning Objectives

Going into this project, I set the following goals for myself:

1. Understand convolutional neural networks (CNNs) - Learn how image classification models work and implement them using PyTorch.
2. Learn transfer learning - Understand how to use pre-trained models like MobileNetV2 and fine-tune them for a specific task.
3. Work with computer vision libraries - Gain experience with OpenCV for image processing and MediaPipe for hand detection.
4. Build a complete application - Go beyond just training a model and create a usable desktop application with a proper GUI.
5. Handle real-time video processing - Learn to process webcam feeds efficiently while maintaining smooth performance.

4. Activities and Tasks

Day 1: Research and Setup

I started by setting up the development environment with Python 3.10, PyTorch, and the necessary libraries. I spent time understanding the problem domain and reviewing papers on gesture recognition. Setting up CUDA for GPU acceleration took some troubleshooting but was essential for training.

Day 2: Data Preparation

I downloaded the ASL Alphabet dataset and wrote scripts to organize and preprocess the images:

- Resizing images to 224x224 pixels (the input size for MobileNetV2)
- Applying data augmentation like rotation, flipping, and brightness adjustments
- Creating training and validation splits
- Building a custom PyTorch Dataset class to load images efficiently

Day 3: Model Development

I experimented with different model architectures:

- Started with a simple custom CNN but accuracy was only around 70%
- Switched to MobileNetV2 with transfer learning and saw immediate improvement
- Also tried EfficientNet-B0 for comparison
- Added dropout layers to prevent overfitting

The final model achieved over 95% validation accuracy after 20 epochs of training.

Day 4: GUI Development

I built the desktop application using Tkinter:

- Created a main window with video display and prediction panels
- Added camera start/stop controls
- Implemented image upload functionality for testing with static images
- Added a history panel to show recent predictions
- Included model loading options and status indicators

Day 5: Integration and Testing

The final phase involved:

- Integrating MediaPipe for hand detection to improve accuracy
- Adding prediction logging functionality
- Testing with different lighting conditions and backgrounds
- Fixing bugs and optimizing performance

5. Skills and Competencies

Through this project, I developed the following technical skills:

Deep Learning:

- Building and training CNNs with PyTorch
- Using transfer learning effectively
- Understanding model evaluation metrics (accuracy, loss, confusion matrices)

Computer Vision:

- Image preprocessing and augmentation
- Real-time video processing with OpenCV
- Hand detection using MediaPipe

Software Development:

- Object-oriented programming in Python
- GUI development with Tkinter
- Project organization and modular code structure
- Using Git for version control

Problem Solving:

- Debugging model training issues
- Optimizing inference speed for real-time performance
- Handling edge cases in gesture recognition

6. Feedback and Evidence

The model evaluation showed strong results:

- Training Accuracy: 97.2%
- Validation Accuracy: 95.8%
- Average Inference Time: 45ms per frame on GPU

During testing with actual sign language users, the system correctly identified most static alphabet signs. The confidence scores provided by the model helped indicate uncertain predictions, and the top-5 prediction display proved useful when the primary prediction was wrong.

The GUI received positive feedback for its clean layout and responsive controls. Users appreciated the ability to switch between camera mode and image upload mode for testing.

7. Challenges and Solutions

Challenge 1: Slow Training on CPU

Initially, training took hours per epoch because CUDA was not configured correctly.

Solution: Reinstalled PyTorch with the correct CUDA version and verified GPU availability. Training time dropped from 4 hours per epoch to about 8 minutes.

Challenge 2: Overfitting on Small Dataset

The model memorized training images but performed poorly on new data.

Solution: Implemented aggressive data augmentation (random rotation, color jitter, horizontal flip) and added dropout layers to the classifier head.

Challenge 3: Poor Performance with Complex Backgrounds

The model gets confused when hands are against cluttered backgrounds.

Solution: Integrated MediaPipe for hand detection to crop the hand region before classification. This focused the model's attention and improved accuracy.

Challenge 4: Threading Issues in GUI

The camera feed would freeze the GUI because video capture was blocking the main thread.

Solution: Moved camera capture to a separate thread and used thread-safe queues to pass frames to the GUI for display.

Challenge 5: Model File Size

The original MobileNetV2 checkpoint was over 100MB, making it inconvenient to share.

Solution: Used state_dict saving instead of saving the entire model, and removed unnecessary optimizer states from checkpoints used for inference only.

8. Outcomes and Impact

The completed Sign Language Detection System achieves the following:

1. Real-time Recognition: Processes webcam input at 20-25 FPS with immediate prediction display.
2. High Accuracy: Over 95% accuracy on validation data for the 29-class ASL alphabet.
3. User-Friendly Interface: Clean desktop application that anyone can use without technical knowledge.
4. Modular Architecture: Code is organized into separate modules for data handling, models, GUI, and utilities.
5. Logging Capability: All predictions are logged with timestamps for later analysis.

The project demonstrates that accessible sign language recognition tools can be built using freely available datasets and open-source libraries. While this system handles static alphabet signs, the architecture could be extended to recognize complete words or phrases with additional training data.

9. Conclusion

This internship project gave me valuable hands-on experience with deep learning and computer vision. Building a complete application from data preparation through model training to final deployment taught me that creating machine learning products involves much more than just training models. The integration work, GUI development, and handling real-world input variability consumed more time than the core machine learning components.

The biggest takeaway was understanding how transfer learning makes complex vision tasks accessible. Using a pre-trained MobileNetV2 backbone allowed me to achieve high accuracy with a relatively small dataset and limited training time. This approach would be my starting point for any future image classification project.

I also gained appreciation for the importance of thorough testing. Many issues only appeared when testing with actual users under realistic conditions. The model that worked perfectly on test images sometimes struggled with webcam input due to differences in lighting, angle, and background.

Going forward, I would like to extend this project to handle dynamic gestures for recognizing words and sentences, not just individual letters. This would require video classification models or recurrent neural networks to capture temporal patterns in hand movements.

Overall, this project was a rewarding experience that strengthened both my technical abilities and my understanding of building practical machine learning applications.

Report submitted as part of internship requirements.
January 2025