Group 17 Progress Report:
Real-Time Sign Language Detection for Accessible Communication

Asher Khan, Hamza Abou Jaib, Mehdi Syed
{khanm406,aboujaih,syedm55}@mcmaster.ca

1 Introduction

Sign language serves as the primary communication method for millions of deaf or hard-of-hearing individuals worldwide. However, accessibility barriers persist, as most non-deaf individuals do not know sign language. Our project aims to build a real-time system that can detect and classify sign language gestures using computer vision and deep learning, translating them into text for easier communication.

The goal is to design a lightweight and accurate system that works in real time without relying on expensive hardware. This task poses challenges due to variations in lighting, backgrounds, hand sizes, and motion between gestures. By optimizing both accuracy and latency, we aim to create a practical model that functions well in uncontrolled environments, allowing inclusive communication across various contexts such as classrooms, workplaces, or healthcare.

2 Related Work

Previous research on sign language recognition demonstrates the effectiveness of deep learning approaches, particularly convolutional and transformer-based models. CS231N (2024) used convolutional neural networks for static ASL gesture classification, achieving high accuracy on the ASL alphabet dataset. Al-Obaidy and Others (2024) explored transformer-based methods for continuous sign recognition, highlighting the trade-off between accuracy and inference speed.

Mobile applications like *SignVision* (Nesar, 2024) have demonstrated the potential of combining CNNs with real-time hand tracking using MediaPipe to achieve efficient on-device recognition. Other studies such as Naz and Khan (2024) have applied transfer learning with MobileNet and ResNet, optimizing for performance on resource-limited devices. Finally, recent work by Rao and Others (2024) showed that lightweight CNNs can achieve competitive results for gesture recognition tasks while maintaining low computational costs, ideal for deployment on edge devices.

We will use these works as references to guide our choice of model architecture, preprocessing strategies, and evaluation metrics for real-time ASL classification.

3 Dataset

The dataset used in this project consists of multiple publicly available **American Sign Language (ASL) image datasets** combined into a single unified corpus. Specifically, the following datasets were used:

1. **American Sign Language Dataset (Train and Test)** from Kaggle (kapillondhe/american-sign-language), containing 166,000 images
2. **ASL Alphabet Dataset** (grassknoted/asl-alphabet), containing 87,000 images.

Each dataset contains images of hands forming letters from the ASL alphabet. The combined dataset was filtered to remove redundant or irrelevant classes (e.g., the "del" class) and normalized for consistent labeling across sources. A custom subclass of `torchvision.datasets.ImageFolder` was implemented to ensure **case-insensitive class detection** and to eliminate duplicate entries (e.g., "Space" vs "space").

After concatenation using `ConcatDataset`, the data was randomly split into:

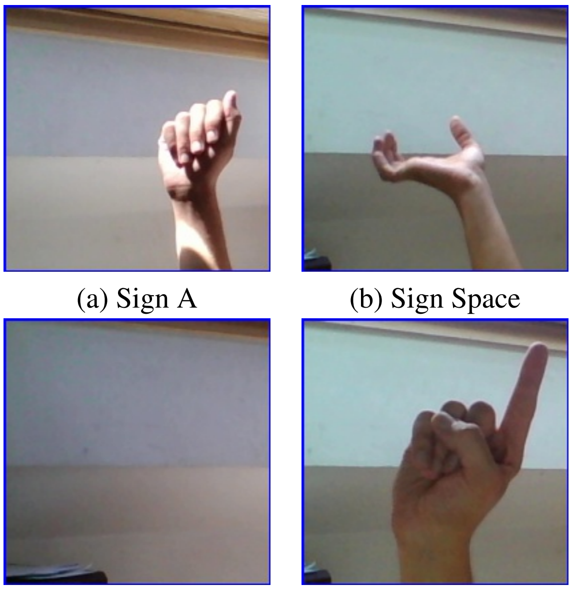
- **80% training set** (199825 Images)
- **10% validation set** (24978 Images)
- **10% test set** (24979 Images)

All images were resized to 224×224 pixels for compatibility with the MobileNetV2 backbone. This combined dataset resulted in a total of approximately **249782 images** spanning **28 distinct sign language classes**, corresponding to the ASL alphabet (A-Z, plus special signs such as "Space" and "Nothing").

To improve generalization and handle variations in lighting, orientation, and hand position, a comprehensive data augmentation pipeline was applied (see Section 4).

4 Features

Our model inputs consisted of JPEG images with resolutions ranging from 200×200 to 400×400 pixels.



(a) Sign A (b) Sign Space

(c) Sign Nothing (d) Sign Z

Table 1: Sample images from the ASL dataset demonstrating variation in gestures, lighting, and background.

Using PyTorch's DataLoader, these images were converted into tensors of shape (3, W, H), where 3 represents the RGB channels. Instead of extracting hand-crafted features, we applied extensive **image preprocessing and augmentation** to improve the model's robustness to real-world variations. These transformations included resizing, normalization, random rotations, affine transformations, color jitter, horizontal flips, and perspective adjustments, ensuring that the model could generalize effectively across diverse hand gestures in all kinds of environments.

4.1 Training Transformations

- **Resizing:** All images scaled to 224×224.
- **RandomRotation ($\pm 10°$):** Simulates small angular differences in hand orientation.
- **RandomAffine (translation $\pm$10%, scale 0.9-1.1, shear=$5°$):** Improves spatial invariance.
- **ColorJitter:** Randomly adjusts brightness, contrast, and saturation to mimic lighting variations.
- **RandomHorizontalFlip (p=0.5):** Accounts for mirrored gestures.
- **RandomPerspective (distortion=0.2, p=0.5):** Adds mild 3D perspective distortions.
- **Normalization:** Standardized each colour channel to ImageNet mean (0.485, 0.456, 0.406) and standard deviation (0.229, 0.224, 0.225).

4.2 Validation/Test Transformations

- **Resizing:** 224×224.
- **Normalization:** Same normalization parameters as above.

Together, these transformations create strong regularization effects, enhancing the network's ability to generalize across subjects and recording environments.

5 Implementation

5.1 Model Architecture

The chosen model is **MobileNetV2**, a lightweight convolutional neural network pre-trained on **ImageNet**.

MobileNetV2 is designed around **depthwise separable convolutions** and **inverted residual blocks**, making it efficient for real-time and embedded applications and ideal for our task.

The final classification layer of the pretrained model was replaced with a new fully connected layer having an output size equal to 28, the number of ASL classes.

$$\mathbf{x}_0 = \text{Input image } (3 \times 224 \times 224) \tag{1}$$

$$\mathbf{x}_1 = \mathrm{ReLU6}(\mathrm{BN}(\mathrm{Conv}_{3\times3}(\mathbf{x}_0))) \tag{2}$$

$$\mathbf{x}_i = \mathrm{InvResBlock}_i(\mathbf{x}_{i-1}), \quad i = 1, \ldots, 17 \tag{3}$$

$$\mathbf{x}_{18} = \mathrm{ReLU6}(\mathrm{BN}(\mathrm{Conv}_{1\times1}(\mathbf{x}_{17}))) \tag{4}$$

$$\mathbf{z} = \mathrm{GlobalAvgPool}(\mathbf{x}_{18}) \tag{5}$$

$$\mathbf{y} = \mathrm{Linear}(\mathrm{Dropout}(\mathbf{z})) \tag{6}$$

Equation 1-6 summarizes the forward pass of our MobileNetV2-based classifier Sandler et al. (2019). The network first applies a standard 3×3 convolution + batch-norm + ReLU6 stem, then processes features through a sequence of inverted residual blocks that use expansion, depthwise convolution, and projection. A final convolution of 1×1 expands the channels to 1280, followed by global average grouping and a dropout + linear classifier. Softmax is applied using the Loss function, instead of wrapping the model output.

Training Configuration

The model was trained in two planned stages (see tables 2 and 3), though only the initial training phase has been completed so far. The second fine-tuning phase will be conducted once the base model converges and stability is verified.

The first stage of training **froze all the convolutional layers** to take advantage of ImageNet features as a strong starting point, while only training the new classifier head.

In the future for the fine-tuning stage, the last five convolutional blocks in the feature extractor will be unfrozen to allow the model to adapt more closely to the ASL domain.

Justification of Key Design Choices:

- **Weight Decay (0.01):** Adds an L_2 penalty to discourage overly large weights, improving generalization and mitigating overfitting.
- **Label Smoothing (0.1):** Prevents the model from becoming overconfident in predictions, improving calibration and robustness to noisy labels.
- **ReduceLROnPlateau Scheduler:** Dynamically lowers the learning rate when validation loss plateaus, ensuring smoother convergence without manual tuning.
- **Adam Optimizer:** Chosen for adaptive learning rate updates and efficient handling of sparse gradients.
- **Two-Stage Strategy:** Allows fast convergence using pretrained ImageNet features, followed by domain-specific adaptation during fine-tuning.

6 Results and Evaluation

Our trained MobileNetV2 model demonstrated strong classification performance on the combined ASL dataset under the given conditions and shows potential for further improvement through additional fine-tuning, with the possibility of achieving results comparable to prior work (see Table 5 for comparisons).

6.1 Quantitative Metrics

Evaluation was performed on the unseen test set using accuracy, precision, recall, and F1-score metrics.

Across the test splits, the model achieved:

Metric	Test (Macro)	Test (Weighted)
Accuracy	87%	87%
Precision	87%	89%
Recall	90%	87%
F1-Score	87%	87%

Table 4: Performance metrics of the model

Macro metrics are calculated as the simple average of each metric (precision, recall, F1-score) across all classes, treating every class equally regardless of its number of samples. This provides a view of model performance that gives equal importance to rare and frequent classes, highlighting how well the model performs on underrepresented classes. In contrast, weighted metrics take class imbalance into account by weighting each class's metric by its number of true samples, reflecting the overall performance on the dataset.

6.2 Training Dynamics

The learning curves show rapid initial improvement, followed by a plateau once early stopping was triggered. This behavior suggests that the relatively high initial learning rate, combined with the scheduler, caused the model to quickly memorize simple patterns in the first epoch. As a result, subsequent epochs showed little improvement, preventing the model from generalizing effectively and stopping short of the 90% accuracy threshold.

Another contributing factor is the training setup during the initial phase. All layers except the classifier head were frozen. This meant that only 35,868 parameters were learnable, while 2,259,740 parameters from the pretrained MobileNetV2 were frozen. This approach leverages pretrained features for quick convergence but limits the model's ability to adapt fully to the ASL dataset.

To address this, the fine-tuning phase will unfreeze the last five convolutional layers, in-

Hyperparameter	Stage 1: Training Configuration
Optimizer	Adam (adaptive learning rate optimizer)
Learning Rate	0.0005 — balances convergence speed and stability
Weight Decay	0.01 — discourages large weights to reduce overfitting
Loss Function	Cross-Entropy with Label Smoothing = 0.1
Scheduler	ReduceLROnPlateau (factor = 0.1, patience = 3)
Batch Size	64 — efficient use of GPU memory and stable gradients
Epochs	5 — initial convergence check and validation
Device	CUDA
Seed	42

Table 2: Hyperparameters used during the initial MobileNetV2 training stage.

Hyperparameter	Stage 2: Planned Fine-Tuning Configuration
Learning Rate	0.00005 — smaller step size for stable fine-tuning
Label Smoothing	0 — allow sharper decision boundaries post-adaptation
Scheduler	ReduceLROnPlateau (factor = 0.5, patience = 2)
Trainable Layers	Last 5 convolutional blocks unfrozen
Epochs	10 (planned)

Table 3: Planned fine-tuning hyperparameters for domain-specific adaptation.

creasing the number of trainable parameters to 1,717,212, while reducing the learning rate heavily. This allows the model to adapt higher-level feature representations to the specific patterns of ASL gestures, while still retaining the ingrained patterns from the pretrained weights, improving generalization and overall accuracy.

6.3 Graphs

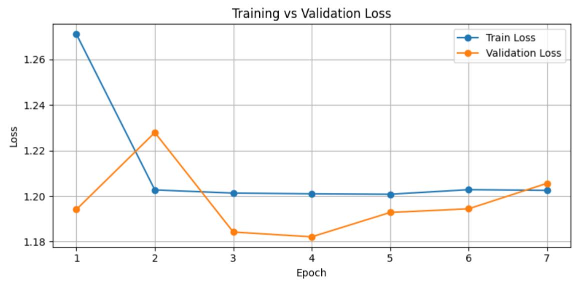


Figure 1: Training and validation loss over epochs.

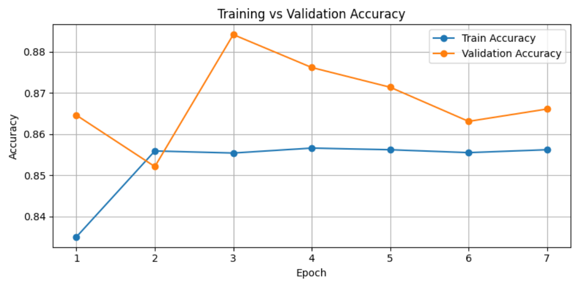


Figure 2: Training and validation accuracy over epochs.

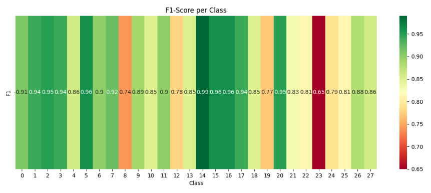


Figure 3: F1 matrix for ASL letter classification.

7 Feedback and Plans

Based on the feedback received from our teaching assistant, we have implemented the following improvements for this project milestone:

- **Update Table 4 Formatting:** The summary table of related works will be revised to follow the *three-line table format* for improved readability.
- **Add Visual Examples of Dataset and Features:** Representative images from the ASL dataset will be included to illustrate sample input gestures and visual variability in lighting, orientation, and background conditions (see table 1).

To continue improving our work, we intend to do the following in the next project milestone:

- **Add a Confusion Matrix for Evaluation:** Include a confusion matrix in our results to provide a more detailed visualization of model performance across different gesture

Study	Model	Dataset	Accuracy (%)
CS231N (2024)	CNN	ASL Finger Spelling	96.6%
Tolentino et al. (2019)	CNN	American Sign Language	93.7%
Abini et al. (2019)	MobileNet-V2	American Sign Language	99.5%
Lum et al. (2020)	MobileNet-V2	ASL Alphabet	98.67%
Pathan et al. (2023)	CNN	ASL Finger Spelling	96.3%
Ours	MobileNet-V2	ASL Alphabet + ASL	87% (Pre-finetuning)

Table 5: Summary of previous studies on sign language recognition and comparison to our model.

classes and identify potential sources of misclassification.

- **Continue Fine-Tuning and Hyperparameter Exploration:** Systematically adjust learning rates, batch sizes, and regularization parameters to optimize model performance and reduce overfitting.
- **Experiment with Layer Freezing Strategies:** Evaluate how varying the number of frozen layers during fine-tuning affects model convergence, feature reuse, and overall classification accuracy.
- **Analyze Class Distribution:** Investigate the number of samples per class to detect any class imbalance issues.

These revisions will enhance the clarity and presentation quality of the report, providing stronger visual and structural support for our methodology.

8 Team Contributions

Asher Khan: Model architecture, transfer learning implementation, and report writing.
Hamza Abou Jaib: Data collection, cleaning, augmentation pipeline, report formatting.
Mehdi Syed: Dataset preprocessing, report writing, result visualization.

References

M.A. Abini, Divya Lakshmi P., K.S. Sharan, and V.N. Sulphiya. 2019. American sign language detection using transfer learning. *International Journal of Computer Trends and Technology*.

A. Al-Obaidy and Others. 2024. Sign language recognition with vision transformers. *ScienceDirect*.

Stanford CS231N. 2024. Sign language recognition with convolutional neural networks. *Course Project Report*.

Kin Yun Lum, Yeh Huann Goh, and Yi Bin Lee. 2020. Asl alphabet recognition using mobilenet-v2. *ASTESJ*.

S. Naz and M. Khan. 2024. American sign language detection using transfer learning. *International Journal of Computer Trends and Technology*.

M. Nesar. 2024. Signvision: A real-time mobile sign language recognition application using chatgpt.

R.K. Pathan, M. Biswas, S. Yasmin, and Others. 2023. Sign language recognition with convolutional neural networks. *Nature Scientific Reports*.

D. Rao and Others. 2024. Lightweight deep learning models for gesture recognition. *SSRN Electronic Journal*.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2019. Mobilenetv2: Inverted residuals and linear bottlenecks. *arXiv preprint arXiv:1801.04381*. Version 4, March 21, 2019.

Lean Karlo Tolentino, Ronnie Serfa Juan, August Thio-ac, Maria Pamahoy, Joni Forteza, and Xavier Garcia. 2019. Static sign language recognition using deep learning. *ResearchGate Preprint*.