Deep Learning-Based American Sign Language Recognition: A Comprehensive Case Study on Sign Language Classification Using Convolutional Neural Networks & ResNet18

Aiden Hock, Bryan Tineo
California State University Long Beach
Profesor Wenlu Zhang

Introduction

American Sign Language (ASL) serves as a crucial communication tool for millions of people. However, the creation of effective automated recognition systems for ASL presents significant challenges. One of the primary hurdles is the considerable variability in hand gestures, which can differ widely between individuals. Additionally, there is a notable scarcity of robust, comprehensive datasets specifically tailored for ASL, which complicates the training and effectiveness of automated systems.

In response to these challenges, this project explores two distinct methodologies for ASL recognition. The first method involves utilizing a pretrained ResNet18 model, which leverages transfer learning to adapt knowledge from a broadly applicable dataset to the specific nuances of ASL. The second approach is the development of a custom convolutional neural network (CNN) from scratch, designed specifically to address the unique characteristics of the ASL dataset used in this study.

Firstly, the project aims to compare the performance of the pretrained ResNet18 model against the custom-built CNN model. This comparison will help ascertain which methodology proves more effective in recognizing ASL signs accurately. Secondly, the study analyzes the trade-offs involved in using transfer learning as opposed to building a recognition system from scratch. This includes evaluating aspects such as the speed of learning, the accuracy of gesture recognition, and the models' ability to generalize from training data to real-world scenarios. Lastly, the project evaluates the different training strategies employed for each model, assessing their impact on the models' performance and efficiency.

Dataset & Related Work

The dataset utilized for this project is the "ayuraj/asl-dataset" available on Kaggle, which includes a collection of 2,515 static hand gesture images spread across 36 classes. These images represent the numbers 0-9 and the letters A-Z in American Sign Language (ASL). The dataset is structured to provide a uniform representation of each class, with most classes containing 70 images. However, one class is represented by only 65 images, introducing a slight imbalance. Each image in the dataset is consistent in size and quality, measuring 400x400 pixels with a resolution of 96 dpi. This standardization helps maintain consistency in training and testing the models.

The dataset used in this project presents three main challenges that could affect the training and performance of the models. First, there is a slight class imbalance, where some classes have more images than others. This could cause the models to perform better on classes with more data and worse on those with less; its small enough to go unnoticed. Second, the dataset is relatively small, with only 70 images per class, this increases the risk of the models overfitting. This means the models could learn too much from the specific images in the training set and not perform well on new, unseen images. Lastly, the images vary in lighting

and the orientation of the hand gestures. While this variability makes the training more challenging by introducing more variations to learn from, it also helps the models become better at generalizing to real-world environments where lighting conditions and orientations are different.

To mitigate these challenges, tailored preprocessing and augmentation strategies were employed to enhance the diversity and robustness of the training dataset. Augmentation techniques such as rotation, scaling, and lighting adjustments were applied to artificially expand the dataset and introduce more variability, thus helping the models learn to generalize better rather than memorize specific images. It's important to note that while these augmentation techniques were applied to the training data, the validation and test datasets were processed without any augmentations. This approach ensures that all models are evaluated on a consistent set of data that closely represents real-world conditions, maintaining the integrity of performance assessments across different model architectures.

Methodology

For this study, ResNet18 was selected due to its proven effectiveness in leveraging transfer learning from extensive datasets like ImageNet, which is beneficial given the small size and high variability of the ASL dataset. Likewise, a scratch-built CNN model was developed to serve as a baseline for comparison and to explore the dataset's unique characteristics independently of any prelearned features.

Both models, were trained under similar configurations to establish a fair baseline for comparison:

- **Learning Rate:** Set at 0.002 to balance effective weight updates with stable convergence.
- **Batch Size:** Maintained at 32 to optimize memory usage while ensuring efficient training cycles.
- **Epochs:** Limited to 10 to achieve model convergence without incurring excessive computational costs.
- **Loss Function:** Cross-entropy loss was utilized for both models, optimizing predictions across the 36 ASL classes.

The process differences between the two models are visually outlined in the table below, which compares the architectural components, input transformations, and specific focus of each model during the training.

	Pretrained Model	Scratch Model
Architecture	ResNet18 pretrained on ImageNet	Sequential Convolutional Neural Network (CNN)
Input Transformation	Pretrained to handle ImageNet-compatible inputs	Flattened 28×28 images into 1D vectors of 784 pixels
Hidden Layers	Not applicable	Two hidden layers with 300 and 100 neurons, using **Rectified Linear Unit (**ReLU) activations
Output Layer	**Fully connected** (FC) layer replaced to output 36 ASL classes	Softmax activation output layer with 36 neurons for each ASL class
Training Focus	Focused solely on training the FC layer	Trained the entire model, starting with raw input feature extraction
Feature Handling	All layers except the FC layer were frozen to retain pretrained features	Learned all features from scratch, starting with raw input data

To ensure a robust evaluation and fair comparison between the models, a 5-fold cross-validation method was employed. This technique is instrumental in reducing bias and variance as it evaluates the models' performance across multiple splits of the dataset, thus providing a comprehensive benchmark of performance. In practice, the dataset was divided into five equal parts or folds, with each model being trained on four folds and validated on the fifth in a

sequential manner. This cycle was repeated until each fold had been used for validation. Throughout this process, key metrics such as accuracy, precision, recall, F1-score, and validation loss were meticulously tracked. The configuration that performed best across the k-fold tests was then selected for additional fine-tuning or to benchmark its performance, ensuring that the most effective model setup was advanced for further evaluation and use.

Dynamic data augmentation was applied across each epoch for every fold. Shared techniques included:

- **Random Rotation:** ±15° to simulate varied hand orientations.
- **Random Horizontal Flip:** 50% probability, introducing bidirectional diversity.
- **Affine Transformations:** Slight translations to increase robustness against positional variability.

Model-specific augmentation details are presented in the table below, showing the unique resizing and normalization for each model architecture.

	Pretrained Model	Scratch Model
Resizing	Resized to **224×224 pixels**, matching ResNet18's input layer.	Resized to **32×32 pixels**, optimized for the scratch model's simpler design.
Normalization	Used **ImageNet statistics**: *mean=[0.485,0.456,0.406], std=[0.229,0.224,0.225].* Ensured compatibility with pretrained weights.	Normalized pixel values to *[−1,1]* using *mean=[0.5,0.5,0.5], std=[0.5,0.5,0.5]* Optimized for custom training.

The scratch-built CNN was designed to learn features comprehensively from scratch, requiring the simultaneous training of all layers from the outset. This involved continuous hyperparameter optimization, including the adjustment of learning rates and other parameters to identify optimal settings for robust training. Additionally, regularization techniques such as dropout were implemented to prevent overfitting and ensure stable learning across all layers.

For the ResNet18, fine-tuning was meticulously executed on the higher convolutional layers, particularly focusing on layers 4 and 3, to tailor the model to the specific needs of the ASL dataset without compromising the integrity of foundational image processing capabilities. Initially, the learning rate was set at 0.002 during the Fully Connected (FC) training phase to establish a broad learning foundation. It was then carefully reduced to 0.0001 for the fine-tuning process to enhance control and stability. The fine-tuning phase was deliberately limited to 5 epochs to refine the model's performance without the risk of overtraining. This process began with the unfreezing of layer 4, where if improvements in performance were noted without signs of overfitting, the fine-tuning extended to layer 3. However, to prevent overfitting, early stopping mechanisms were employed to halt training if an increase in validation loss was observed, reverting the model to its last best-performing configuration.

Experimental Setup

The project used Python for machine learning workflows and Jupyter Notebook for documentation. PyTorch was the primary deep learning framework for its flexibility and ease of use. Torchvision facilitated image preprocessing and the use of pretrained models, enhancing training efficiency. Scikit-learn was used for statistical analysis, including generating confusion matrices and classification reports to evaluate model performance. Matplotlib was employed for plotting training metrics and visualizations to assess performance throughout the research.

All computational tasks were carried out on Google Colab, which provided access to high-performance GPU resources, via the cloud. This setup was essential for managing the computationally demanding aspects of deep learning models without the need for local hardware resources, allowing for scalable and efficient model training. Collaboration was streamlined through the use of a shared Google Drive folder, making scripts and datasets easily accessible to all team members. Additionally, version control and project history management were handled through Git, with the repository hosted on GitHub to maintain a clear record of the project's development.

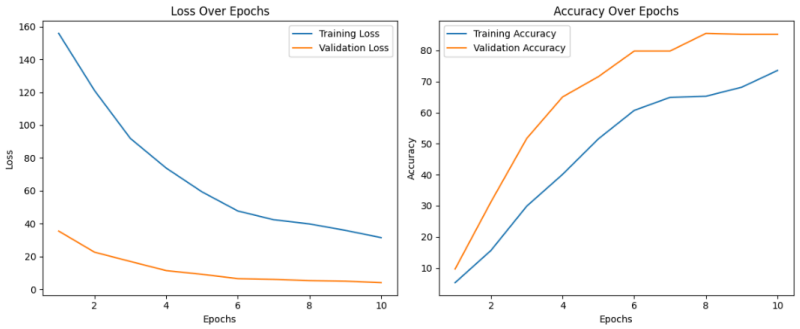
Measurement

To evaluate the ASL classification models effectively, we utilized key metrics including accuracy, precision, recall, F1-score, and confusion matrices, implemented through PyTorch and enhanced with scikit-learn for deeper statistical analysis. Accuracy provided a quick measure of overall performance, but precision and recall were critical for assessing the model's reliability and its ability to correctly identify ASL signs. The F1-score was especially valuable for understanding the balance between precision and recall, particularly in the context of potential class imbalances. Confusion matrices helped identify specific misclassifications, offering targeted insights for model refinement. These metrics played a crucial role in benchmarking and analyzing learning convergence throughout the training and validation phases. During fine-tuning, careful attention was paid to validation loss to ensure that adjustments were made to prevent overfitting. This approach allowed for dynamic model optimization. Through these thorough evaluations, each model's strengths and areas for refinement were clearly identified, enabling precise adjustments to optimize learning and generalization.

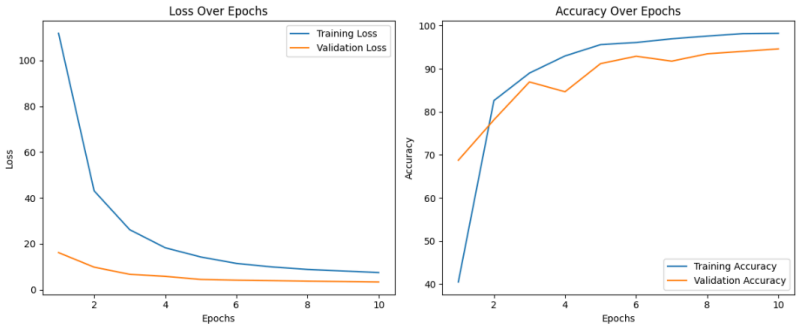
Result Analysis, Intuitions, and Comparison

Both models showcased significant improvements over their training periods, but the pretrained model particularly excelled in both efficiency and effectiveness. It demonstrated a steeper and more rapid decline in both training and validation loss, as depicted in the *Loss Over Epochs graph*, suggesting a higher level of initial tuning and quicker adaptation to the training data. Concurrently, this model achieved superior accuracy at a faster rate compared to the scratch-built model. This accelerated improvement and notably higher final accuracy are clearly illustrated in the *Accuracy Over Epochs graph*.

Scratch Model

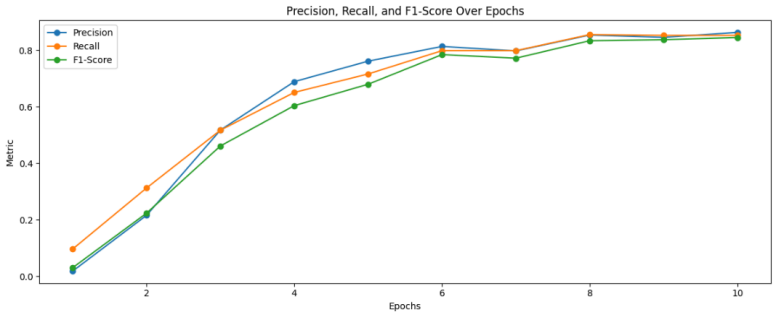


Pretrained Model

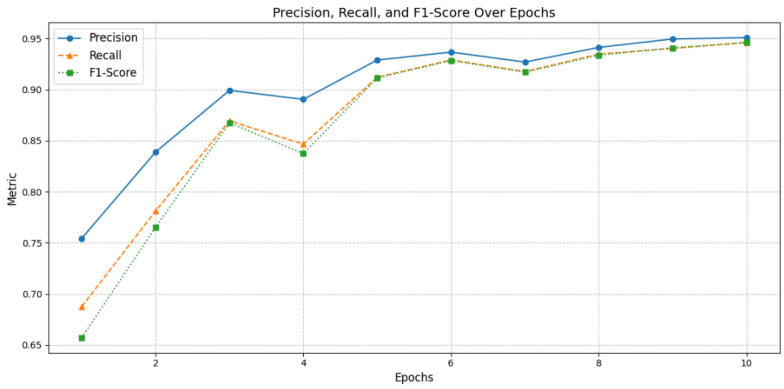


Precision, Recall, and F1-Score Dynamics
The pretrained model consistently outperformed the scratch model across the metrics of precision, recall, and F1-score. This superiority in performance and stability is clearly illustrated in the visual representation provided by the *Precision, Recall, and F1-Score Over Epochs graph.*

Scratch Model

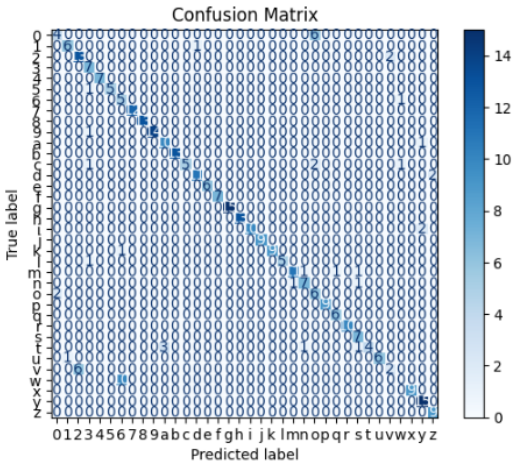


Pretrained Model

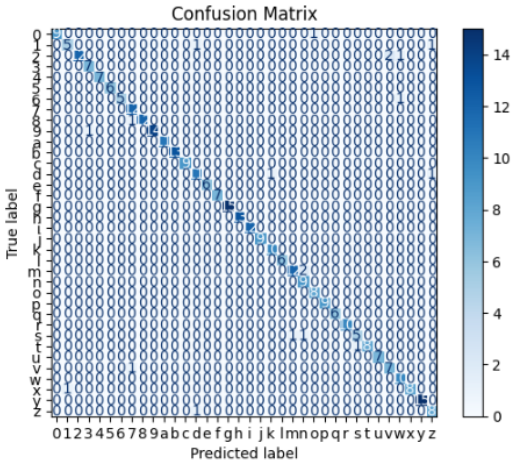


The confusion matrices for both models provide a detailed view of class-specific performance. In these matrices, the pretrained model demonstrated fewer misclassifications, showcasing its superior accuracy, as depicted in the *Confusion Matrix* visuals.

Scratch Model



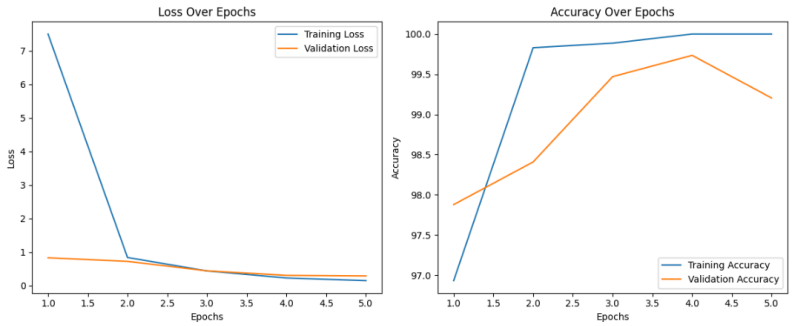
Pretrained Model



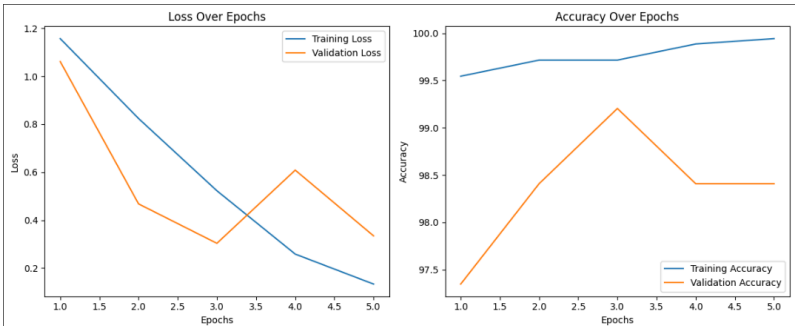
Initially, fine-tuning layer 4 of the pretrained model led to optimal performance enhancements across all evaluated metrics. This phase of fine-tuning is clearly illustrated in the *Loss and Accuracy Over Epochs during Layer 4 Fine-Tuning graph*, which shows marked improvements in both training and validation metrics. However, subsequent fine-tuning of layer 3 initially seemed promising but ultimately resulted in overfitting. This was evidenced by an increase in validation loss and a decline in validation accuracy starting after the third

epoch, as detailed in the *Loss and Accuracy Over Epochs during Layer 3 Fine-Tuning graph*.

Layer 4



Layer 3



Final Model Results Comparison Table

Metric	Scratch Model	Pretrained Model	Pretrained Fine-Tuned Model
Training Loss	31.4586	7.4860	0.1443
Validation Loss	4.0248	3.3352	0.2861
Training Accuracy (%)	73.58	98.22	100.00
Validation Accuracy (%)	85.23	94.60	99.20
Precision	0.86	0.95	0.99
Recall	0.85	0.95	0.99
F1-Score	0.84	0.95	0.99

As shown in the *Final Model Results Comparison Table* The scratch model achieved a validation accuracy of 85.23% and a training accuracy of 73.58%, indicating underfitting but good generalization to unseen data. The pretrained model reached a peak validation accuracy of 94.60%, with training and validation accuracies closely aligned, showing the benefits of transfer learning. The fine-tuned pretrained model achieved near-perfect training accuracy (100%) and a validation accuracy of 99.20%, demonstrating improved generalization after fine-tuning. The slight reduction in validation accuracy towards the end indicates the onset of overfitting, highlighting the need for careful monitoring during extended training

Conclusion

In our project on ASL recognition, we conducted a comparative analysis between a pretrained ResNet18 model and a scratch-built CNN. The findings clearly demonstrate that the pretrained ResNet18 model outperformed the scratch-built model in terms of learning efficiency and generalization capabilities. During the fine-tuning of layer 3 in the ResNet18 model, we observed overfitting by the third epoch, indicated by an increase in validation loss, which suggested the model was memorizing rather than generalizing the training data. In contrast, the scratch-built model, while initially underfitting and displaying lower accuracy on the training data, showed impressive generalization to new data, indicating its robustness despite simpler learned features.

Despite these nuances, the pretrained ResNet18 model emerged as the clear

winner in our study. It consistently achieved higher performance metrics and demonstrated significant adaptability, albeit with signs of overfitting towards the end of the training process. This behavior underlines the challenges of finely tuning a model that has been pretrained on a vast and diverse dataset like ImageNet to specific tasks such as ASL recognition.

The superior performance of the pretrained model underscores the effectiveness of transfer learning, particularly in scenarios involving limited data and high variability, typical of specialized datasets like ASL. This advantage suggests that future efforts in ASL recognition should lean towards optimizing pretrained models, and applying careful fine-tuning strategies to maximize their potential without compromising their ability to generalize. The success of the pretrained model not only highlights its immediate utility in enhancing technological accessibility for the ASL community but also points to broader implications for deploying deep learning in real-world applications. By refining our approach to the application of pretrained models, we can significantly improve the performance and usability of technologies that assist in ASL communication, thereby fostering greater inclusion and accessibility for users who rely on sign language.

Code Contributions

Aiden Hock

ML_ASL_fromScratch.ipynb
- Initialize Scratch Model

PreTrainedASLClassificationModel.ipynb
- Pulled and separated initialized pretrained model from SignLanguageClassificationProject and completed the pretrained model in this file

Bryan Tineo

SignLanguageClassificationProject.ipynb
- Initialize Pretrained Model & Completed Scratch model

ScratchASLClassificationModel.ipynb
- Separated completed scratch model into this file from SignLanguageClassificationProject

Relevant Sources

1. GitHub repository https://github.com/bryanmax9/SignLangClassifier

2. Kaggle ASL dataset https://www.kaggle.com/ayuraj/asl-dataset