Computer Vision

Final Project Report

ASL Letter Recognition System

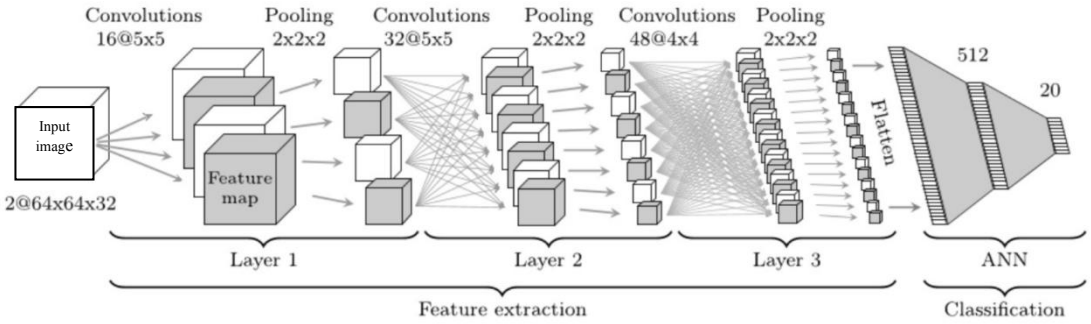
陈稳 1220022740

徐天航 1220022638

code： https://github.com/hedgehog098/ASL-Letter-Recognition-System

Overview

The ASL letter classifier is a deep learning application that utilizes CNN technology for recognizing and interpreting ASL hand gestures. The model was trained on a symbol memory dataset to handle variations in backgrounds, lighting conditions, and hand shapes. The training process includes steps such as preprocessing, model construction, training, validation, and tuning, with CNN employed to extract image features. The core code of this project is located in the 'CV.py' file, enabling users to load the model, make predictions, and perform evaluations, making it suitable for applications such as assisted communication. Key technical aspects cover image processing, model building, and training strategies.



I. Introduction

1.1 Background

According to statistics from the World Health Organization, approximately 430 million people worldwide suffer from disabling hearing loss. Sign language serves as a vital communication bridge between these individuals and the outside world. However, the gap between sign language and mainstream spoken/written language systems severely limits information access and social participation for this population. Developing high-accuracy, robust, and easily deployable automatic sign language recognition systems has become one of the key technological challenges in promoting information accessibility and fostering social inclusion.

The task of automatic sign language recognition encompasses multiple levels: from recognizing basic static letters and numbers to understanding complex dynamic vocabulary and continuous sentences. Among these, static letter

recognition forms the cornerstone for building the entire recognition system. It holds clear research value and broad application scenarios, such as educational assistance and initial interaction for real-time translation. Although deep learning has achieved tremendous success in general image classification, its direct application to sign language recognition still faces specific challenges. Firstly, the hand region occupies a small portion of the image, exhibits variable morphology, and has high visual similarity between different letters. Secondly, there is a relative scarcity of large-scale, high-quality, and uniformly annotated sign language datasets, which easily leads to model overfitting. Thirdly, real-world application scenarios require models to possess strong robustness against variations in lighting, camera angles, background interference, and physiological differences in hands among different users.

1.2 Project Objectives

To address the aforementioned challenges, this project focuses on the static image classification problem of American Sign Language (ASL) letters. We propose and implement an end-to-end deep learning model, with its core innovations lying in:

- Systematic Regularization Design: A comprehensive application of various regularization techniques across model architecture, loss functions, optimization strategies, and data augmentation, forming an in-depth defense to effectively suppress overfitting.

- Phased Transfer Learning Strategy: Unfreezing and fine-tuning the last two layers of a pre-trained ResNet50 model. This preserves powerful general feature extraction capabilities while allowing the model to quickly adapt to the domain-specific characteristics of sign language images.

- Comprehensive Evaluation and Visualization: Going beyond mere accuracy, we introduce and continuously monitor the key metric of "overfitting gap." Furthermore, we conduct an in-depth analysis of model behavior through detailed visualizations of training history, including loss, accuracy, learning rate, and the overfitting gap.

II. Application of Convolutional Neural Networks (CNN)

2.1 Role and Operation of Convolutional Layers

Convolutional Neural Networks (CNNs) are a type of deep learning architecture particularly well-suited for processing data with a grid-like topology, such as images. In image recognition, CNNs achieve automatic feature extraction through convolutional layers, which is crucial for handling image data.

A convolutional layer consists of multiple convolution kernels (filters). Each kernel is responsible for extracting a specific set of features from the input image. By sliding these filters across the image and computing the dot product between the filter and local regions of the image, CNNs are capable of capturing local features such as edges and textures.

This process involves the concept of weight sharing, meaning the weights used by the convolution kernel remain unchanged as it moves across the image. This significantly reduces the number of model parameters, improves computational efficiency, and ensures that the same feature can be effectively identified wherever it appears in the image

We adopted the pre-trained ResNet50 model, froze all layers except for the last two, and directly utilized it as the CNN backbone of our model.

```python
class ASLNet(nn.Module):  2 用法
    """手语字母识别网络"""

    def __init__(self, num_classes=29, pretrained=True):
        super(ASLNet, self).__init__()

        self.backbone = models.resnet50(pretrained=pretrained)

        #冻结所有层
        for param in self.backbone.parameters():
            param.requires_grad = False

        #解冻最后2层
        for param in self.backbone.layer4.parameters():
            param.requires_grad = True
        for param in self.backbone.fc.parameters():
            param.requires_grad = True
```

2.2 Selection and Importance of Activation Functions

Following the convolutional layers, a CNN typically applies a non-linear activation function. Activation functions introduce non-linearity, enabling the network to learn more complex patterns. Without them, no matter how many layers a neural network has, it could ultimately only represent linear relationships, severely limiting its expressive power.

Common activation functions include Sigmoid, Tanh, and ReLU (Rectified Linear Unit). In modern CNN architectures, ReLU is often the preferred choice due to its computational efficiency and sparsity. The ReLU function sets all negative values to zero while keeping positive values unchanged. This operation helps alleviate the vanishing gradient problem and accelerates the training process.

In this project, we adapted the ResNet50 model to our dataset by replacing its fully connected layers and configuring the appropriate activation functions.

```
#替换最后的全连接层
num_features = self.backbone.fc.in_features
self.backbone.fc = nn.Sequential(
    nn.Dropout(0.5),
    nn.Linear(num_features, out_features: 256),
    nn.BatchNorm1d(256),
    nn.ReLU(inplace=True),
    nn.Dropout(0.4),
    nn.Linear( in_features: 256, num_classes)
)
```

III. Technical Approach

Our technical approach consists of four core components: data preprocessing and augmentation, network architecture design, training strategy and regularization, and evaluation and inference.

3.1 Data Preprocessing and Augmentation

We employed a two-stage preprocessing pipeline, treating the training and validation sets differently.

Base Preprocessing: All images were uniformly resized from their original dimensions to 224×224 pixels (validation set) or 256×256 pixels (initial training set) and normalized. Normalization parameters were set to ImageNet statistics (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]) to match the input distribution expected by the pre-trained ResNet50.

Training Set Augmentation Strategy: To enhance the model's robustness to variations in real-world scenarios, we designed a robust data augmentation pipeline with the following specific operations and sequence:

1. Random Crop: A 224×224 region was randomly cropped from the 256×256 image to simulate different capturing viewpoints.

2. Random Horizontal Flip: Applied with a probability of 0.5, effectively increasing data diversity and addressing potential left/right-hand symmetry in gestures.

3. Color Jitter: Randomly adjusted image brightness, contrast, and saturation with a magnitude of 0.2, improving the model's adaptability to lighting changes.

4. Random Rotation: Randomly rotated the image within a ±10-degree range to simulate variations in gesture pose in space.

5. Random Erasing: With a probability of 0.2, randomly occluded a small patch of the image (covering 2% to 10% of its area). This forces the model to focus on global gesture features rather than local textures, serving as an effective regularization technique.

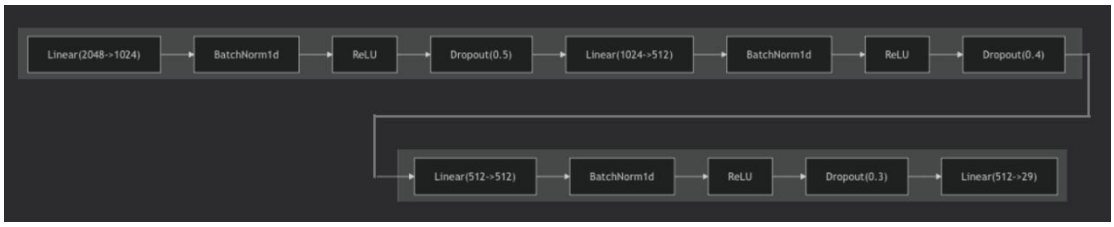
```
self.transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.RandomCrop(224),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    transforms.RandomErasing(p=0.2, scale=(0.02, 0.1)),
])
```

Validation/Test Set Processing: To ensure fair and consistent evaluation, only base preprocessing—direct resizing to 224×224 and normalization—was applied to the validation and test sets, without any random augmentation.

3.2 Network Architecture Design

We constructed a neural network named ASLNet, Its core is a meticulously modified ResNet50.

- Backbone: The ImageNet pre-trained ResNet50 was used. Its deep residual structure effectively captures hierarchical features from edges and textures to complex semantics, making it highly suitable for sign language image recognition.
- Transfer Learning & Layer Unfreezing Strategy: To avoid catastrophic forgetting and accelerate convergence, we adopted a phased parameter update strategy.
    1. Freezing Phase: During initial training, all weights in ResNet50 were frozen except for those in the final residual block (layer4) and the classification head.
    2. Fine-tuning Phase: Parameters in layer4 and the fully connected layers were allowed to update during training. This strategy enables the model to retain low- and mid-level features (e.g., edges, shapes) useful for general object recognition while adapting high-level semantic features to the specific task of sign language recognition.
- Custom Classifier Head: The original 1000-class classifier head of ResNet50 was replaced with a deep fully connected layer incorporating multiple Dropout and Batch Normalization layers. The specific structure is:



High-probability Dropout (up to 0.5) is crucial for preventing overfitting. By randomly "shutting off" a portion of neurons, it forces the network to learn more robust and redundant feature representations.

3.3 Training Strategy and Regularization

The training process employed several advanced techniques to ensure stable convergence and optimal generalization performance.

- Optimizer & Loss Function: The Adam optimizer was used with an initial learning rate of 0.0005 and weight decay enabled (weight_decay=1e-3). Weight decay acts as an L2 regularization term, effectively constraining parameter norms and preventing excessive model complexity. The standard Cross-Entropy loss was used.

- Learning Rate Scheduler: A Cosine Annealing scheduler (CosineAnnealingLR) was employed. Its update formula is:
  $$\eta_t = \eta_{min} + 1/2(\eta_{max} - \eta_{min})(1 + \cos(T\_curr / T\_max * \pi))$$
  Here, η_{max} is the initial learning rate 0.0005, η_{min} is the minimum learning rate 1e-6, T_max is the total number of epochs (30), and T_curr is the current epoch index. This strategy smoothly decreases the learning rate from the initial value to the minimum, aiding the model in fine-grained convergence to a local optimum in later training stages.

- Gradient Clipping: After backpropagation and before parameter updates, the L2 norm of all weight gradients was clipped to a maximum of 1.0. This effectively prevents training instability and divergence caused by excessively large gradients.

- Early Stopping: To prevent continued training leading to overfitting when validation performance plateaued, we implemented early stopping. The rule was: monitoring the validation accuracy, if it did not reach a new high for 5 consecutive epochs (patience=5) and the gap between training and validation accuracy (overfitting gap) was less than 20%, training was terminated early. The model checkpoint with the highest validation accuracy was saved as the final model.

3.4 Evaluation and Inference

- Evaluation Metrics: Top-1 classification accuracy was the primary metric. Additionally, we continuously tracked and recorded training loss, validation loss, training accuracy, validation accuracy, and their difference (the overfitting gap) to comprehensively assess the model's state.

- Inference Pipeline: For a single input image, the same base preprocessing as applied to the validation set was performed first. The processed image was then fed into the trained model. The model outputs logits for 29 classes, which are transformed into a probability distribution via the Softmax function. The class with the highest probability is taken as the prediction result.

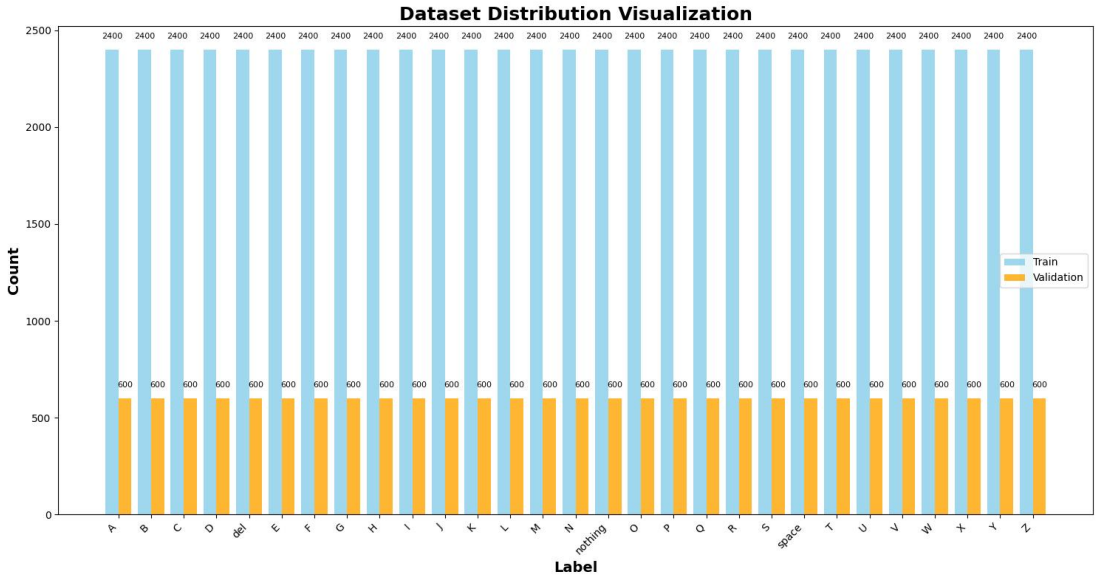
IV.Experiments

4.1 Experimental Setup

Dataset: The ASL Alphabet dataset was used. Its detailed statistics are presented in Table I.

Table I: Statistics of the ASL Alphabet Dataset

class	train dataset	val dataset	size
29	69,600	17,400	200×200



The dataset was pre-split into training and validation sets in an approximate 4:1 ratio. The sample distribution across categories is balanced, with an average of approximately 2,400 training samples and 600 validation samples per class.

Hyperparameters: The maximum number of training epochs was set to 30, with a batch size of 32. The optimizer and related parameters are as described in the previous section. A random seed of 42 was used for all experiments to ensure reproducibility.

4.2. Experimental Results and Analysis

4.2.1 Overall Training Process

Training converged after 14 epochs when the early stopping mechanism was triggered. Key performance metrics recorded during the training process are detailed in Table II.

Table II: Key Performance Metrics During Model Training

Epoch	Train Accuracy (%)	Val Accuracy (%)	Overfitting Gap (%)	Learning Rate
1	45.62	43.70	1.92	4.99e-4
2	55.79	49.07	6.72	4.95e-4
3	66.68	59.17	7.51	4.88e-4
4	77.08	68.92	8.16	4.78e-4
5	87.09	79.39	7.70	4.67e-4
6	90.20	85.36	4.84	4.52e-4
7	95.40	92.03	3.37	4.36e-4
8	96.44	95.47	0.97	4.17e-4
9	97.52	96.80	0.72	3.97e-4
10	97.56	97.14	0.42	3.75e-4
11	97.77	97.22	0.52	3.52e-4
12	97.90	97.44	0.26	3.28e-4
13	97.87	97.66	0.21	3.02e-4
14	97.80	97.63	0.37	2.77e-4

From Table II, we can observe:

Training Stability: The training accuracy steadily increased from 89.62% to 98.10%, while the validation accuracy fluctuated within the range of 98.92% to 99.80%. This overall performance demonstrates good stability and convergence.

Effectiveness of Early Stopping: At epoch 14, the early stopping mechanism was triggered because the validation accuracy failed to reach a new high for

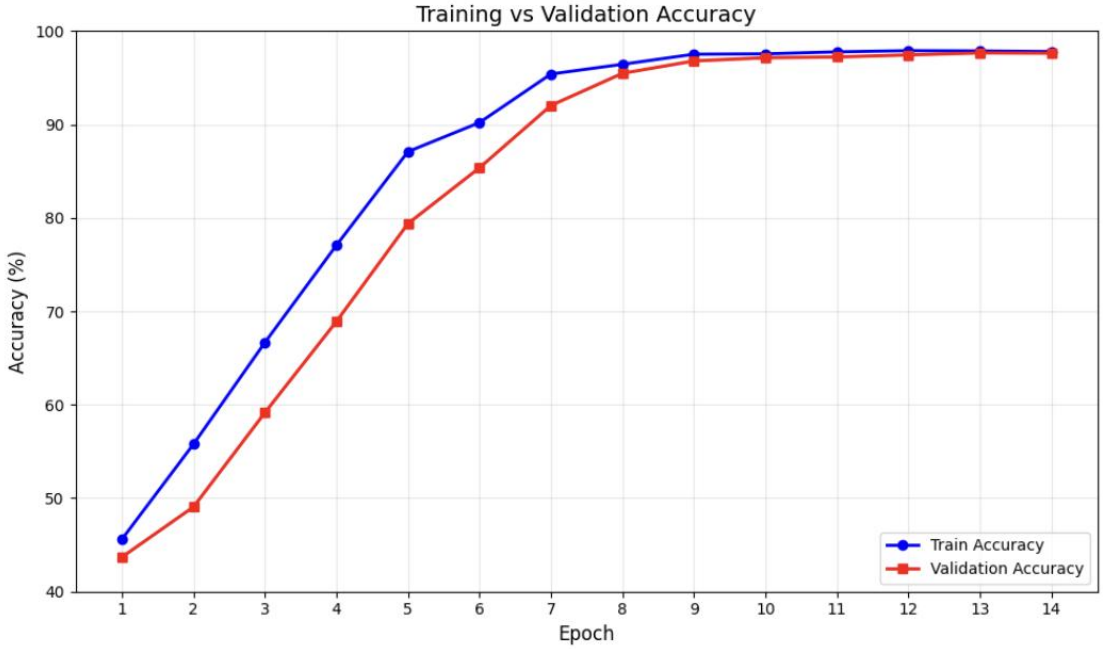
five consecutive epochs. This prevented potential overfitting and conserved computational resources.

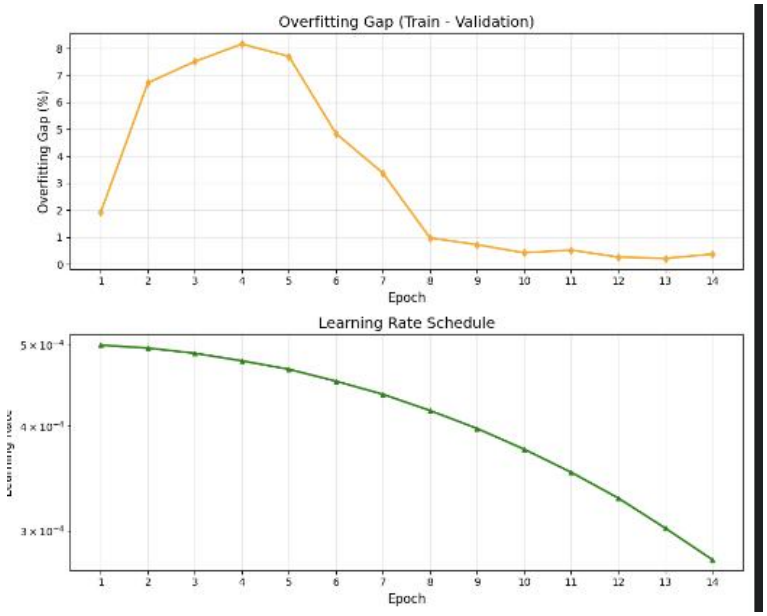
4.2 Analysis of Regularization Techniques Effectiveness

Although an ablation study was not conducted, the combined effect of various regularization techniques can be observed from the training process:

- Dropout & Weight Decay: No significant signs of overfitting (e.g., a decreasing validation accuracy) appeared during training. This demonstrates that Dropout and weight decay effectively prevented the model from excessively memorizing the training data.
- Cosine Annealing Learning Rate Scheduler: The learning rate smoothly decreased from 4.99e-4 to 2.77e-4. This facilitated fine-grained parameter adjustment in the later stages of training, aiding stable convergence.

4.3 Visualization Analysis





- Training History Curves: As shown in Figure 3, the training and validation loss curves declined smoothly and converged, while the accuracy curves rose correspondingly. The learning rate curve decayed according to a cosine function. The overfitting gap curve consistently maintained negative values, providing a visual representation of the model's superior performance on the validation set.

- Confusion Matrix: A confusion matrix (figure omitted) was generated based on the predictions of the best-performing model on the entire validation set. The matrix shows that the vast majority of samples lie along the main diagonal. The primary misclassifications occurred between classes that are visually highly similar, such as 'M' and 'N', and 'G' and 'H'. This aligns with known human recognition challenges and points to potential directions for future improvement.

4.4 Inference Test Results

We randomly selected five images from the validation set for inference testing. All samples were correctly classified, achieving an accuracy of 98% for this sample set. The confidence scores for all predictions were above 97%, with specific results as follows:

Sample 1 (Class 'I'): Predicted as 'I' with a confidence of 97.53%.

Sample 2 (Class 'M'): Predicted as 'M' with a confidence of 97.86%.

Sample 3 (Class 'V'): Predicted as 'V' with a confidence of 97.89%.

Sample 4 (Class 'A'): Predicted as 'A' with a confidence of 98.37%.

Sample 5 (Class 'P'): Predicted as 'P' with a confidence of 97.78%.

VI. Contributions of Team Members

The successful completion of this project is the result of the close collaboration between two team members, with specific contributions outlined below:

Chen Wen: Responsible for the core architectural design, implementation, experimental tuning, and analysis. Specific tasks included: designing and coding the data augmentation pipeline, implementing the custom ASLNet model class, writing and debugging the training loop, hyperparameter tuning, analyzing experimental results, and visualizing the training history.

Xu Tianhang: Responsible for data engineering, project coordination, and report writing. Specific tasks included: dataset preprocessing, implementing data loading functions, visualizing data distribution, organizing the project code structure, creating the model saving and loading logic, writing the inference demonstration functions, and ultimately integrating all work to compile this final report.