ECE371 Neural Networks and Deep Learning Assignment 1

Hu Chenran 22308050
School of Electronics and Communication Engineering
Sun Yat-sen University, Shenzhen Campus
huchr5@mail2.sysu.edu.cn

Abstract: This assignment tackles the problem of flower image classification using a convolutional neural network. By completing a partial PyTorch implementation, we apply data augmentation, modify the ResNet-18 architecture, and adopt an SGD-based training strategy. Experimental results show that our model achieves a peak validation accuracy of 96.49%, demonstrating the effectiveness of the chosen techniques. We also analyze training stability and model generalization.

Keywords: Image Classification, Deep Learning, ResNet, PyTorch

1 Introduction

In this assignment, we address the problem of image classification using a deep convolutional neural network. The task involves classifying images from a flower dataset into their respective categories. The dataset is processed and trained using the ResNet-18 model, a widely used architecture for image classification. Our implementation focuses on completing a partially provided PyTorch script by filling in crucial components such as data augmentation, model modification, loss function definition, optimization strategy, training loop, and model saving. The final model achieves high accuracy on the validation dataset and is saved automatically when the best validation accuracy is reached.

2 Related Work

Image classification has been a core problem in computer vision for decades. Early approaches relied heavily on handcrafted features and traditional machine learning classifiers. With the advent of deep learning, convolutional neural networks (CNNs) have revolutionized the field.

One of the earliest successful CNN architectures was LeNet-5, proposed by LeCun et al. [1]. Later, architectures such as AlexNet [2] and VGGNet [3] demonstrated that deeper networks can achieve higher accuracy. ResNet, introduced by He et al. [4], further improved performance by introducing residual connections that help in training very deep networks by mitigating the vanishing gradient problem. ResNet-18, a relatively shallow variant, remains a popular choice for small- to medium-scale image classification tasks due to its balance between computational efficiency and accuracy.

3 Method

We started with a partially completed PyTorch script and completed all the graded sections. The major steps are outlined below:

- **Data Augmentation**: We applied multiple transformations such as rotation, cropping, horizontal and vertical flipping, color jittering, resizing, and normalization. This increases the diversity of training samples and helps prevent overfitting.

- **Model Modification**: We loaded the pre-trained ResNet-18 model and replaced the final fully connected layer to match the number of classes in our flower dataset. A dropout layer was also added before the final layer to enhance generalization.
- **Loss Function and Optimizer**: Cross-entropy loss was chosen as the criterion, which is standard for multi-class classification. SGD with momentum, weight decay, and Nesterov acceleration was used for optimization.
- **Training Loop**: A training loop was implemented to iterate over epochs, compute training and validation loss and accuracy, and update the learning rate using a scheduler. The model was evaluated on the validation set after each epoch, and the best model was saved.

4 Experiments

Dataset and Environment

The flower dataset was split into 80% for training and 20% for validation. All experiments were run on a GPU-enabled environment using PyTorch.

Data Preprocessing and Augmentation Strategy

In this experiment, we designed two separate data preprocessing pipelines for the training and validation datasets, as shown in the code below:

Listing 1: Data augmentation and preprocessing pipeline for training and validation datasets

```
data_transforms1 = transforms.Compose([
    transforms.RandomRotation(30),
    transforms.CenterCrop(448),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomVerticalFlip(p=0.5),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
    transforms.ToTensor(),
    transforms.Resize(224, antialias=True),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

data_transforms2 = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

The transformation pipeline `data_transforms1` is applied to the training dataset. It includes five data augmentation techniques: random rotation by up to 30 degrees, center cropping to size 448, random horizontal and vertical flips with 50% probability, and color jittering that slightly adjusts brightness, contrast, saturation, and hue. These augmentations aim to introduce diversity into the dataset and simulate various visual scenarios, thereby reducing the risk of overfitting and improving the model's generalization capabilities. After augmentation, images are converted to tensors, resized to 224 (to match input size requirements of common CNNs), and normalized using ImageNet-standard mean and standard deviation.

On the other hand, the validation dataset uses `data_transforms2`, which applies only deterministic preprocessing steps: resizing to 256, center cropping to 224, tensor conversion, and normalization. These transformations ensure that the evaluation of model performance is conducted on data with consistent appearance and scale, avoiding the randomness of augmentation which could affect the fairness and repeatability of validation results.

This separation of training and validation preprocessing pipelines follows best practices in deep learning, where augmentations are introduced only during training to improve robustness, while validation remains clean and standardized for accurate performance measurement.

4.1 Dropout Regularization Before the Linear Layer

To mitigate overfitting in the fully connected layers, a `Dropout` layer was added before the final `Linear` layer. Dropout is a regularization technique that randomly sets a fraction of input units to zero during training, which prevents the co-adaptation of neurons and forces the network to learn more robust and distributed representations. This is particularly important in deep convolutional networks, where the final dense layers are often prone to overfitting due to their large number of parameters. By introducing dropout, we aim to improve the generalization performance of the model on unseen data [5].

4.2 Optimization Strategy Using SGD with Nesterov Momentum

For optimization, we employed Stochastic Gradient Descent (SGD) with a learning rate of 0.001, a weight decay of 5×10^{-4}, momentum of 0.9, and Nesterov acceleration enabled. This configuration was chosen based on its proven effectiveness in training deep neural networks. The use of momentum helps accelerate convergence by dampening oscillations in the optimization trajectory, especially in ravines of the loss surface. Nesterov momentum further improves this by looking ahead at the approximate future position, which often results in faster and more stable convergence [6]. Additionally, weight decay introduces L2 regularization, which penalizes large weights and helps reduce overfitting. Together, these settings provide a good balance between convergence speed and generalization.

Performance Metrics

We tracked the loss, accuracy, and learning rate for each epoch. Below is a sample output:

```
Epoch 0/24
----------
Learning Rate: 0.001000
train Loss: 0.7942 Acc: 0.6888
val Loss: 0.2127 Acc: 0.9333
...
Epoch 24/24
----------
Learning Rate: 0.000001
train Loss: 0.0408 Acc: 0.9908
val Loss: 0.1008 Acc: 0.9614

Training complete in 13m 33s
Best val Acc: 0.964912
```

Model Saving

The best performing model (based on validation accuracy) was saved to the *Ex2\ work_dir* directory as *best_model.pth*.

5 Discussion

The model's training and validation performance indicates a well-generalized learning process. As shown in the training logs, the validation accuracy improved consistently from 68.88% in the first epoch to a peak of 96.49% at epoch 17, while the validation loss steadily decreased, reaching a

minimum of 0.1008. These results suggest that the model effectively learned discriminative features from the data.

Several design choices contributed to this outcome. First, the incorporation of a `Dropout` layer before the final `Linear` classifier helped reduce overfitting by randomly deactivating neurons during training. This regularization strategy forces the model to learn more robust representations that generalize better to unseen data.

Second, the optimizer used was SGD with momentum and Nesterov acceleration, along with weight decay. This configuration provided both stability and faster convergence, especially in the early stages of training, by smoothing out oscillations and encouraging movement in more predictive directions.

Third, learning rate scheduling was essential. We used a multi-step scheduler to reduce the learning rate at epochs 7 and 14. This allowed the model to make large weight updates initially, followed by more refined updates during later epochs, which helped converge to a better local minimum. This staged reduction in learning rate likely contributed to the gradual and stable improvement in accuracy, especially during epochs 7–17.

Finally, minor fluctuations in validation loss—particularly between epochs 8 and 13—can be attributed to the stochastic nature of data augmentation and batch sampling. Nevertheless, these variations were small and did not affect the overall upward trend in validation accuracy.

5.1 Future Work

To further enhance performance, several directions can be explored in future work. First, testing deeper network architectures such as ResNet-50 or DenseNet may allow the model to capture more complex feature hierarchies. Second, alternative optimizers like Adam or RMSprop could be evaluated for their adaptive learning rate behavior. Third, more sophisticated data augmentation techniques—such as CutMix, MixUp, or AutoAugment—may improve the diversity of training data and enhance generalization.

Moreover, fine-tuning hyperparameters such as batch size, dropout rate, and learning rate schedule could yield further improvements. Incorporating techniques like label smoothing or knowledge distillation could also be beneficial, particularly in tasks with limited data.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[3] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[5] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

[6] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning (ICML)*, pages 1139–1147, 2013.