ECE371 Neural Networks and Deep Learning Assignment 1

Xin Zeng 22308007
School of Electronics and Communication Engineering
Sun Yat-sen University, Shenzhen Campus
zengx79@mail2.sysu.edu.cn

Abstract: In this assignment, I fine-tuned a pre-trained ResNet18 model using PyTorch for five-class flower classification. I implemented data augmentation, modified the final fully connected layer to output five classes, and used cross-entropy loss with the Adam optimizer. Backpropagation and model saving were also completed. The model achieved over 80% accuracy on the validation set. This work shows that transfer learning is effective for small-scale image classification tasks.

Keywords: PyTorch, Transfer Learning, Image Classification

1 Introduction

This project addresses the problem of multi-class image classification of flowers, specifically five categories: daisy (588 images), dandelion (556), rose (583), sunflower (536), and tulip (585). To solve this, a ResNet18 model from `torchvision` with pre-trained ImageNet weights was fine-tuned. The final fully connected layer was modified to output five classes corresponding to the flower types.

My main conclusions are as follows. First, data augmentation techniques such as random horizontal flipping, resizing, and normalization significantly improved model generalization by reducing overfitting on the relatively small dataset. Second, using the Adam optimizer yielded faster convergence and better performance than SGD in this setting. Input normalization to match the ImageNet mean and standard deviation was essential for achieving reasonable performance, due to the use of ImageNet pre-trained weights. However, the overall accuracy remained just above 80%, likely limited by class imbalance, the small dataset size, and intra-class visual similarity among flowers.

2 Related Work

Image classification has been a central problem in computer vision for decades. Early approaches relied on hand-crafted features such as SIFT and HOG, combined with traditional classifiers like support vector machines [1]. The landscape changed significantly with the success of deep convolutional neural networks (CNNs), particularly after the introduction of AlexNet [2], which demonstrated the power of deep learning on large-scale datasets like ImageNet.

Subsequent models such as VGGNet [3], GoogLeNet [4], and ResNet [5] introduced deeper architectures and novel modules like inception and residual connections, allowing networks to scale while mitigating issues such as vanishing gradients. ResNet in particular marked a significant milestone due to its ease of optimization and superior performance, and it remains a widely adopted backbone for transfer learning tasks, including fine-tuning on smaller datasets.

More recent trends include the integration of attention mechanisms, such as in SENet [6], and the emergence of transformer-based models like Vision Transformer (ViT) [7], which move away from convolutional structures altogether. However, for small to medium-sized datasets, fine-tuning pre-

trained CNNs such as ResNet remains a robust and computationally efficient choice, especially when data is limited.

In this work, I adopt ResNet18 due to its balance between model complexity and performance, leveraging its pre-trained ImageNet weights to perform fine-tuning on a five-class flower classification task.

3 Method

My implementation consists of five main components: data augmentation, model modification, loss function, optimizer selection, and training with backpropagation and model saving.

Data augmentation. To improve generalization and reduce overfitting, I applied a series of data augmentation techniques during training. These include `RandomResizedCrop` for scale-invariant learning, `RandomHorizontalFlip` and `RandomVerticalFlip` for spatial diversity, `RandomRotation` to increase rotational robustness, and `ColorJitter` to simulate lighting and color variations. Finally, all images were normalized using the ImageNet mean and standard deviation to match the pre-training conditions of ResNet18.

Model modification. I adopted the ResNet18 model from `torchvision` with default ImageNet weights. To adapt the network to our five-class flower classification task, the final fully connected layer was replaced with a new `Linear` layer with five output neurons. This adjustment enables the model to output class logits for the specific categories in our dataset.

Loss function. I used cross-entropy loss (`nn.CrossEntropyLoss`) as it is well-suited for multi-class classification problems and aligns with the softmax-based output of the modified model.

Optimizer. The Adam optimizer (`optim.Adam`) was selected with a learning rate of 1×10^{-4} for its adaptive learning rate mechanism, which generally results in faster convergence compared to vanilla SGD, particularly on small to medium datasets like flower dataset.

Training and model saving. During each iteration, gradients were computed via backpropagation (`loss.backward()`), followed by parameter updates using the optimizer. The model achieving the best validation accuracy during training was saved to disk using `torch.save()` for later evaluation or deployment.

4 Experiments

This section presents a comprehensive evaluation of the model under different settings, focusing on three main aspects: data augmentation, optimizer selection, and learning rate variation. I report both quantitative results and qualitative analysis to highlight the effects of these factors on classification performance.

4.1 Experiment 1: Effects of Data Augmentation

To evaluate the role of data augmentation, I designed five settings:

- **Full augmentation (baseline)**: includes random resized cropping, horizontal and vertical flipping, rotation, color jitter, normalization.
- **No normalization**: removes the normalization step.
- **No flip**: removes both horizontal and vertical flip.
- **No rotation and color jitter**: removes both rotation and color jitter components.
- **No augmentation**: only resizing and conversion to tensor.

Table 1: Validation Accuracy under Different Data Augmentation Strategies

Augmentation Strategy	Validation Accuracy (%)
Full augmentation	**82.28**
No normalization	**75.26**
No flip	**76.32**
No rotation and color jitter	**78.42**
No augmentation	**73.68**

All experiments in this section are conducted under a consistent training configuration: the Adam optimizer with a learning rate of 1×10^{-4}, and a total of 10 training epochs.

The results show that full augmentation consistently outperformed the ablated variants. In particular, the absence of normalization leads to a significant performance drop, highlighting its importance when using pre-trained ImageNet models. Removing flip or rotation/color jitter caused mild decreases in performance, confirming that geometric and color diversity helps generalization. Without any augmentation, the model suffered from overfitting and failed to generalize well.

4.2 Experiment 2: Optimizer Comparison

I compared four popular optimizers using the same initial learning rate (1×10^{-4}) and other hyperparameters:

- **Adam (baseline)**: An adaptive learning rate optimization algorithm that combines momentum and RMS-based updates. It typically performs well in practice with minimal tuning.
- **SGD with momentum**: A classic optimizer that updates weights using gradients and accumulates a velocity term to accelerate convergence and avoid local minima.
- **RMSProp**: Uses an exponentially decaying average of squared gradients to adjust learning rates, making it suitable for non-stationary objectives.
- **Adagrad**: Adjusts the learning rate for each parameter individually based on the historical magnitude of gradients, which can be beneficial for sparse data.

Table 2: Validation Accuracy under Different Optimizers

Optimizer	Validation Accuracy (%)
Adam	**82.28**
SGD	**78.07**
RMSProp	**78.25**
Adagrad	**79.24**

All experiments in this section are conducted under a consistent training configuration: full data augmentation with a learning rate of 1×10^{-4}, and a total of 10 training epochs.

Adam achieved the best accuracy among all tested optimizers, likely due to its adaptive learning rate and momentum handling. SGD performed reasonably well but required more epochs to converge. RMSProp was unstable in the early epochs, while Adagrad struggled with vanishing updates due to its aggressive learning rate decay.

4.3 Experiment 3: Impact of Learning Rate

To analyze sensitivity to learning rate, I trained the model with different values: 1×10^{-2}, 1×10^{-3}, 1×10^{-4} (baseline), and 1×10^{-5}, using Adam optimizer. Validation accuracy and training loss were tracked over epochs.

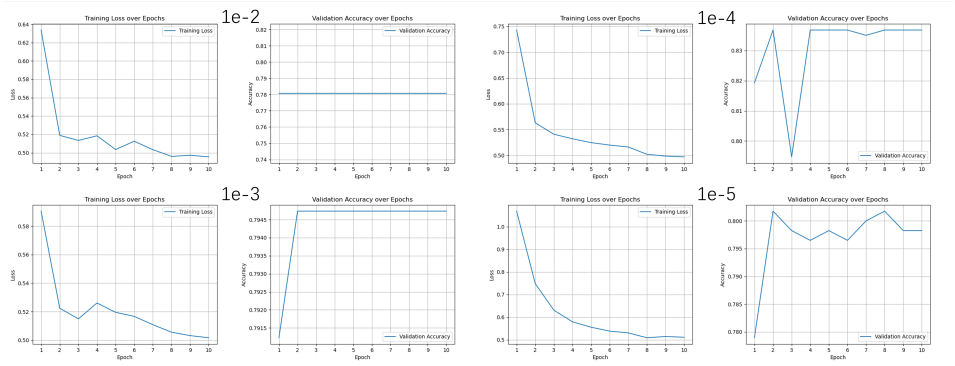

Figure 1: Training loss and validation accuracy for different learning rates

As shown in Figure 1, we compare the training loss and validation accuracy across four different learning rates arranged in a 2×4 grid. A learning rate of 1×10^{-4} yielded the best balance between convergence speed and stability. Higher learning rates (e.g., 1×10^{-2}) led to unstable training and oscillation, while lower values like 1×10^{-5} resulted in slow convergence and suboptimal performance.

4.4 Discussion

From these experiments, I draw the following conclusions:

- Data augmentation, especially normalization and geometric/color transformations, plays a crucial role in improving model generalization.
- Adam optimizer is well-suited for small datasets due to its adaptive nature.
- Learning rate tuning is essential: overly aggressive values can destabilize training, while conservative ones slow down convergence.

These findings reinforce classroom discussions on the importance of inductive bias (via augmentation), optimization stability, and learning rate schedules in deep learning training.

References

[1] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *CVPR*, 2005.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *NeurIPS*, 2012.

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

[4] C. Szegedy et al. Going deeper with convolutions. In *CVPR*, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CVPR*, 2016.

[6] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. *CVPR*, 2018.

[7] A. Dosovitskiy et al. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.