ECE371 Neural Networks and Deep Learning Assignment 1

musheng Fan
School of Electronics and Communication Engineering
Sun Yat-sen University, Shenzhen Campus
fanmsh3@mail2.sysu.edu.cn

Abstract: This experiment completed the ECE371 Assignment 1 on flower classification, covering model fine - tuning with MMClassification and a PyTorch training script. The dataset was split and organized in ImageNet format. By fine - tuning a pre - trained model and enhancing the training script, the model achieved good validation accuracy, demonstrating deep learning's effectiveness in image classification.

Keywords: MMClassification flower classification

1 Introduction

This assignment aims to train models for flower classification. In Exercise 1, I learn how to finish a Fine-tune classification model using MMClassification. In Exercise 2, I Complete the classification model by training script.

2 Related Work

Simonyan and Zisserman [1] experimentally demonstrated that network depth enhances model performance. Their VGGNet, with its simple, regular convolutional and pooling layers, advanced deep network exploration but had high computational costs and overfitting tendencies.

He et al. [2] introduced ResNet, which uses residual connections to ease the vanishing gradient problem in deep networks. This enabled the construction of networks with hundreds to thousands of layers, boosting performance and offering valuable insights for future model design.

3 Method

Firstly,I learn how to install the environment and how to use github to git clone mmpretrain.

for the environment: Step 1 Download and install Miniconda from the official website. Step 2 Create a conda virtual environment and activate it.

```
conda create --name openmmlab python=3.8 -y
conda activate openmmlab
```

for the mmpretrain:

```
git clone https://github.com/open-mmlab/mmpretrain.git
cd mmpretrain
pip install -U openmim && mim install -e
```

then we start to pretrain we use model of resnet in mmpretrain
we copy resnet50_8xb32_in1k.py as resnet50_8xb32-ft_custom.py

, .

and then we write in it to reload the configaration before reload the configaration, I organize the dataset of flower to follow the stantard of imagenet.

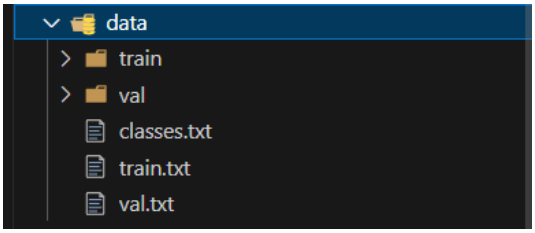

Figure 1: Convert the dataset into the ImageNet format

After that,I begin to finish the configaration and train

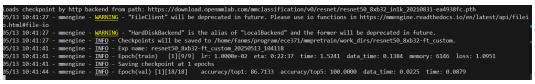

Figure 2: train

Secondly,I learn how to finish the code block to classify the flower dataset

I add Rotation,flip and colorjitter in dataset for training. then I resize them into 244x244 and add normalization

Figure 3: data processing

```
num_classes = len(class_names)                    #
model.fc = nn.Linear(model.fc.in_features, num_classes)
```

Then I add a linear layer to change the num_classes output of the model.

Figure 4: Enter Caption

Finally i define the loss function,optimizer and learning rate scheduler for train.

```
loss.backward()
optimizer.step()
```

In training, We use loss.backward() to calculate the gradient and upgrade the parameter by gradient.

When the training end,We save the model by torch.save to save model.state_dict()

4 Experiments

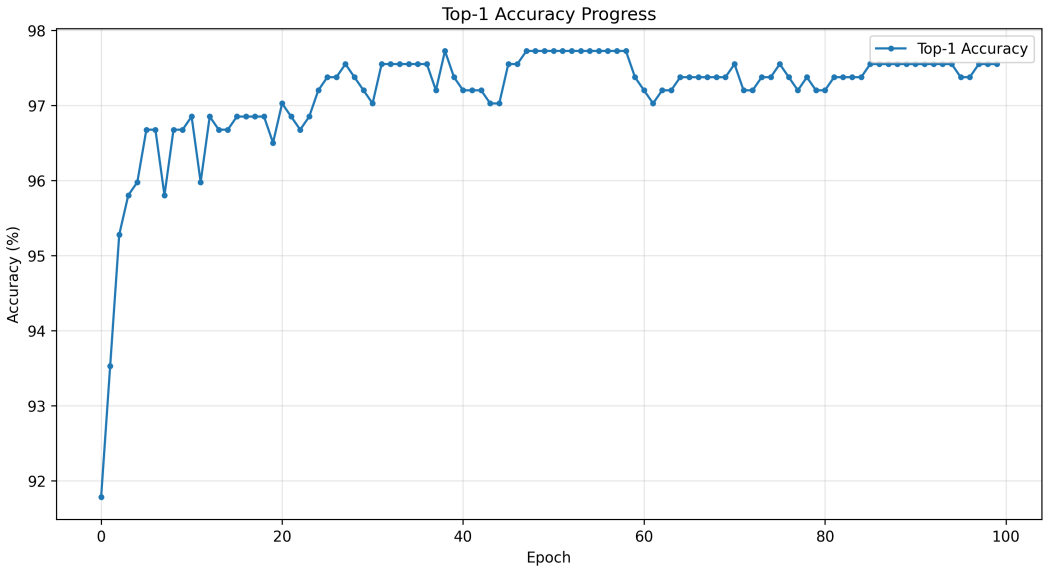

Figure 5: mmpretrain

Accuracy surges from 91.8% to approximately 95.5% within the first 5 epochs, indicating strong feature learning enabled by the backbone network (likely a pretrained model fine-tuned with high initial learning rates).

It stabilizes to approximately 97.5% by Epoch 20, suggesting efficient adaptation of the classification head to the target dataset

Let's mainly look about the exercise 2

In exercise 2 I firstly use SGD optimizer to train

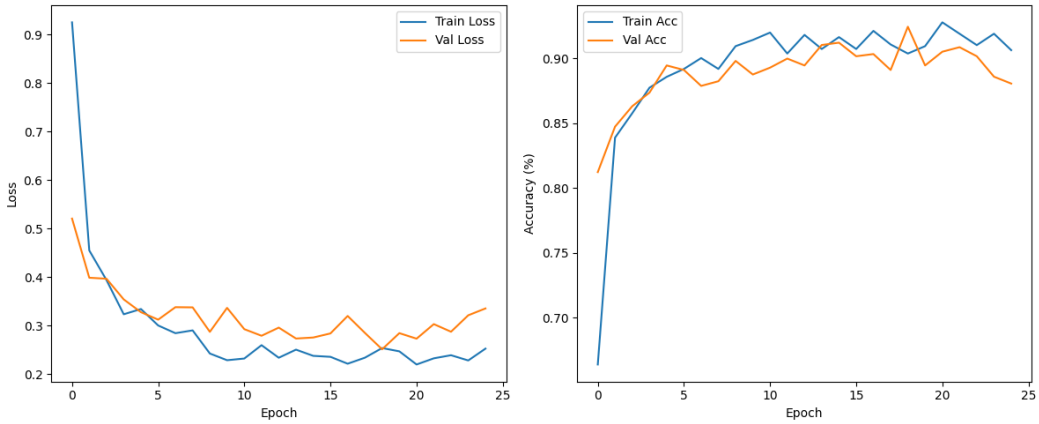

Figure 6: SGD

then I change SGD to adam

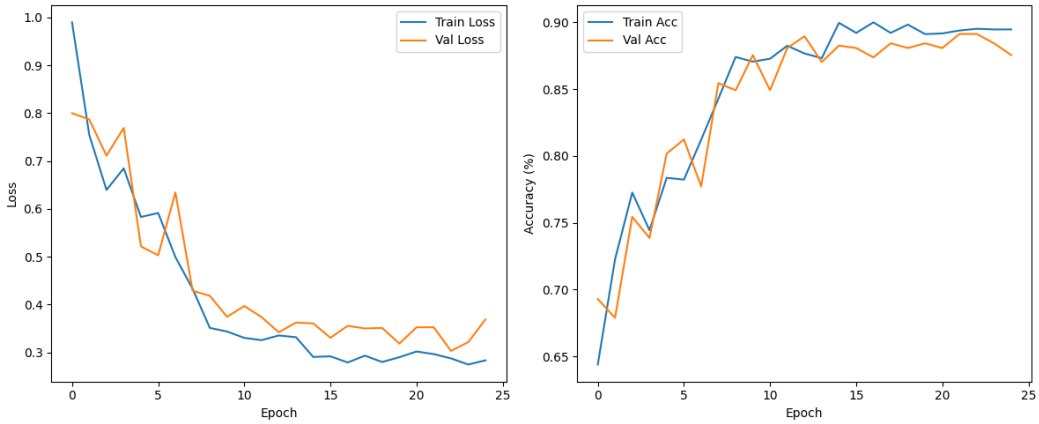

Figure 7: adam

we can contrast between SGD and Adam

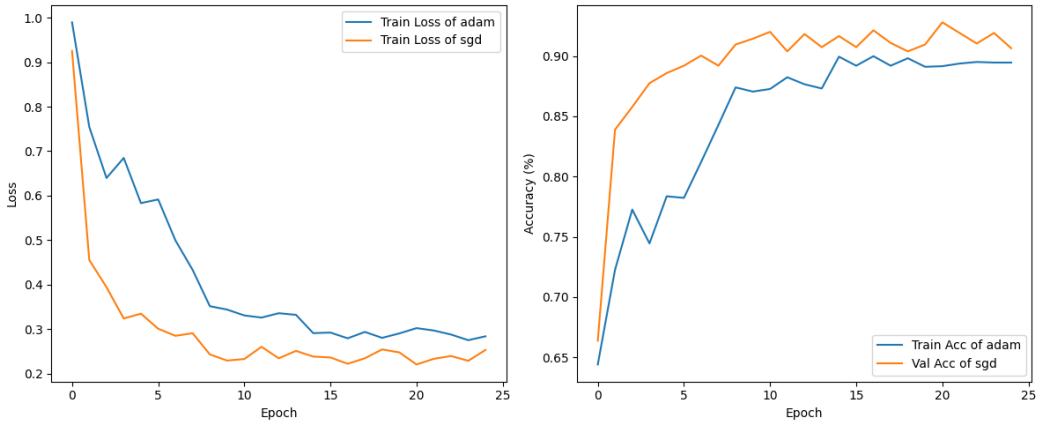

Figure 8: contrast

When i learn about the differences between Sgd and Adam, I guess the reasons. Adam adjusts the learning rate adaptively through the first and second moments of the gradient. This mechanism can lead to significant fluctuations in the learning rate during the initial training phase, causing instability in loss and accuracy. In my experiment, Adam's loss curve exhibited notable fluctuations at the beginning, whereas SGDś loss curve was relatively smooth. What's more, our data is simple, so the SGD maybe more suitable for our train.

then i try to change the batchsize from 32 to 64

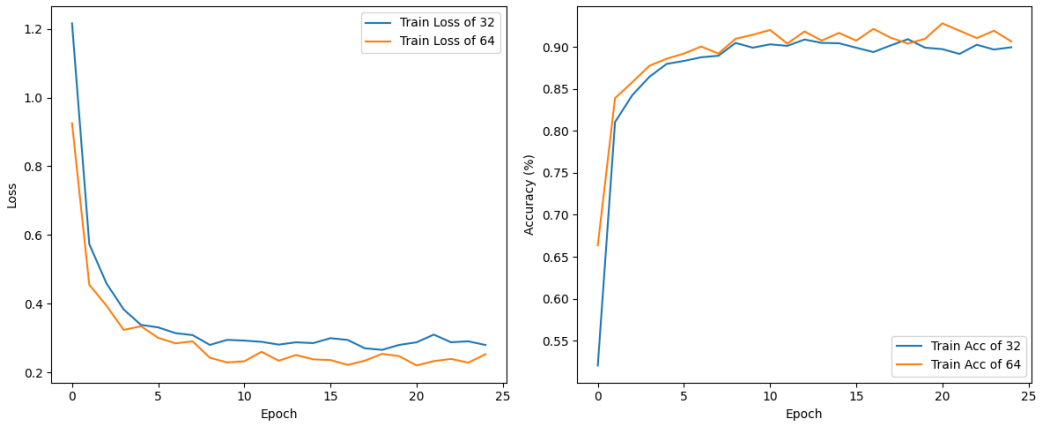

Figure 9: batch

At the early stage of training (from epoch 0 to around epoch 5), the training loss with batch size 32 decreases faster than that with batch size 64. This indicates that the model with batch size 32 adjusts its parameters more effectively in the initial phase, enabling it to start learning the data patterns more quickly. As training progresses, the training loss with batch size 64 gradually catches up and eventually becomes slightly lower than that with batch size 32. This suggests that in the later stages of training, a larger batch size helps the model converge more stably to a lower loss value. This may be because a larger batch size provides a more accurate estimation of the gradient across the entire dataset, reducing the random fluctuations during training and making the parameter updates more stable, thereby achieving a lower loss.

The training accuracy with batch size 64 rises more rapidly from the beginning and reaches a higher level in the early stages of training but finally both of them are almost the same

References

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

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.