Flower classification based on ResNet18

Ning Yang

May 12, 2025

Abstract

Using flower_dataset to train ResNet18 on OpenMMLab and PyTorch respectively and completed the classification task. The two models achieved accuracies of 91.4% and 95.4% on the validation set.

1 Exercise 1

1.1 Preparation of dataset

Flower_dataset contains a total of five categories of flower data, which are Daisy, Dandelion, Rose, Sunflower, and Tulip. The data for each type of flower is placed separately in a folder. In order to make the dataset conform to the ImageNet format, we need to perform some processing on it.

I used a Python script to help me handle it (See Figure 1). This script can automatically divide the original dataset into a training set and a validation set in an 8: 2 ratio, and can also automatically generate a mapping text file of categories and file names for each set.

```python
cur_path = os.getcwd()
orig_path = os.path.join(cur_path, 'flower_dataset')
targ_path = os.path.join(cur_path, 'dataset')
train_path = os.path.join(targ_path, 'train')
val_path = os.path.join(targ_path, 'val')

os.mkdir(train_path)
os.mkdir(val_path)

Classes_file = open(targ_path+'\\classes.txt', 'w')
Train_file = open(targ_path+'\\train.txt', 'w')
Val_file = open(targ_path+'\\val.txt', 'w')

Classes = os.listdir(orig_path)
for i, clas in tqdm(enumerate(Classes)):
    orig_class_path = os.path.join(orig_path, clas)
    targ_train_class_path = os.path.join(train_path, clas)
    targ_val_class_path = os.path.join(val_path, clas)
    os.mkdir(targ_train_class_path)
    os.mkdir(targ_val_class_path)
    Classes_file.write(clas+'\n')
    Samples = os.listdir(orig_class_path)
    for j, sample in enumerate(Samples):
        orig_data_path = os.path.join(orig_class_path, sample)
        if j%5 == 0:
            targ_data_path = os.path.join(targ_val_class_path, sample)
            Val_file.write(clas+'/'+sample+' '+str(i)+'\n')
        else:
            targ_data_path = os.path.join(targ_train_class_path, sample)
            Train_file.write(clas+'/'+sample+' '+str(i)+'\n')
        shutil.copy(orig_data_path, targ_data_path)

Classes_file.close()
Train_file.close()
Val_file.close()
```

Figure 1: Part of Python script

After running the script, we can see that the data has been correctly partitioned, and the corresponding text files have also been generated.

Class	Total	Train	Val
Daisy	588	470	118
Dandelion	556	444	112
Rose	583	466	117
Sunflower	536	428	108
Tulip	585	468	117

Table 1: Number of images in the dataset.

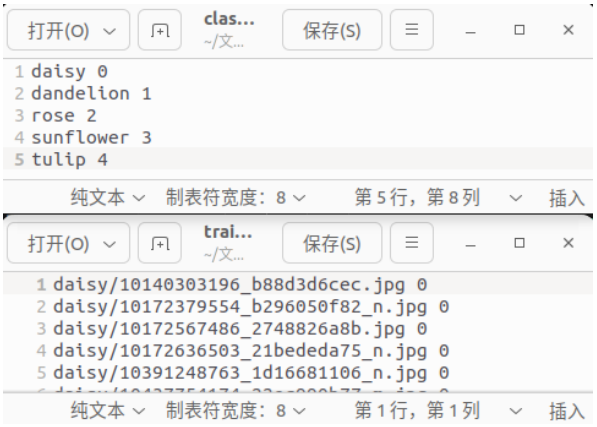

Figure 2: Automatically generated text file

1.2 Generate open-mmlab configuration file

Since we need to use ResNet18 for fine-tuning, we can find the corresponding configuration file in mmclassification/config/resnet. Here we choose to use the configuration of resnet18_8xb32_in1k.py. To facilitate changing all configuration items at once, we first copy the path of the configuration script and then execute:

python mmclassification/tools/train.py mmclassification/configs/resnet/resnet18_8xb32_in1k.py

The terminal will report error messages, but don't worry, what we need is a complete configuration file that it automatically generates for us. This file will be saved in a folder with the same name as the configuration file we used under work-dirs.

Most of the information in the configuration file is already complete. We just need to supplement the paths of our training and validation sets, adjust the number of classes below to fine tune the model. In addition, in order to achieve better training results, I set the decrease in learning rate to be slower and increased the training epochs to 1000; By default, the top 1 and top 5 accuracy rates are returned. Since we only have five classes, we have changed top 5 to top 3; Due to my machine having 16GB of video memory, in order to speed up the training process, I have increased the batch size to 256.

After modifying the configuration file, save it as a new configuration file.

1.3 Model fine-tuning

Copy the modified configuration file path and replace the configuration file path portion of the instructions in 2.2 with the copied path. Execute this command, and the training will automatically proceed. We can view the real-time training status of the model through the output of the terminal.

After training, we can determine the model with the highest accuracy based on the output of the terminal. Since I have set the model to be saved once per epoch, I can directly retain the model with the highest accuracy. Its accuracy on the validation set is 91.4%.(Figure 3)

```
2025/04/28 09:36:18 - mmengine - INFO - Exp name: my_resnet18_20250428_093251
2025/04/28 09:36:18 - mmengine - INFO - Epoch(train)  [47][72/72]  lr: 1.0000e-03
eta: 0:03:10  time: 0.0411  data_time: 0.0001  memory: 878  loss: 0.1139
2025/04/28 09:36:18 - mmengine - INFO - Saving checkpoint at 47 epochs
2025/04/28 09:36:19 - mmengine - INFO - Epoch(val) [47][18/18]    accuracy/top1:
91.4336  accuracy/top3: 99.1259  data_time: 0.0104  time: 0.0259
2025/04/28 09:36:23 - mmengine - INFO - Exp name: my_resnet18_20250428_093251
2025/04/28 09:36:23 - mmengine - INFO - Epoch(train)  [48][72/72]  lr: 1.0000e-03
eta: 0:03:06  time: 0.0411  data_time: 0.0001  memory: 878  loss: 0.2105
2025/04/28 09:36:23 - mmengine - INFO - Saving checkpoint at 48 epochs
```

Figure 3: Best accuracy in logs

2 Exercise 2

2.1 Preparation of dataset

In Exercise 1, they were randomly divided into a training set and a validation set in an 8: 2 ratio. However, it can be found in the provided training script that the script has already implemented the function of randomly dividing the dataset, so it is necessary to restore the dataset to its original state.

2.2 Complete training script

In the data augmentation methods section, I added five methods including RandomResizedCrop, RandomHorizontalFlip, RandomRotation, ColorJitter and RandomVerticalFlip. They will randomly crop, rotate, horizontally and vertically flip the image, and randomly change parameters such as brightness and contrast of the image. Then I used cross entropy loss as the loss function, Adam as the optimizer, and set the learning rate reduction strategy to decrease by 0.1 times every 10 epochs.

In addition, I slightly modified the original training function to enable the script to save logs of various parameters during the training process. This can facilitate the visualization of the training process later.

2.3 Training and Results

In the first training session, I set batch size to 256 and trained for 100 epochs. The model ultimately achieved an best accuracy of 91.3% on the validation set. The loss and accuracy curves of the training are shown in Figure 4.

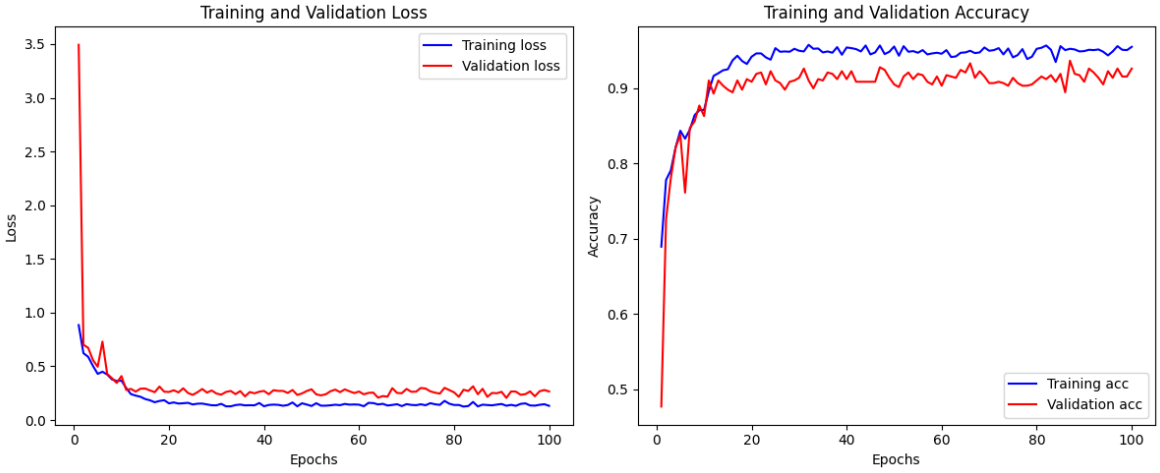

Figure 4: Loss and accuracy curves

In order to further improve the accuracy of the model, I re optimized the learning rate strategy and retrained it based on the weights obtained from the first training. In the second training session, I set the learning rate reduction strategy to decrease by 0.5 times every 20 steps and trained for 400

epochs. The best accuracy of the model on the validation set reached 95.4%, which is slightly higher than the accuracy of the first model. The loss and accuracy curves during the training process are shown in Figure 5.

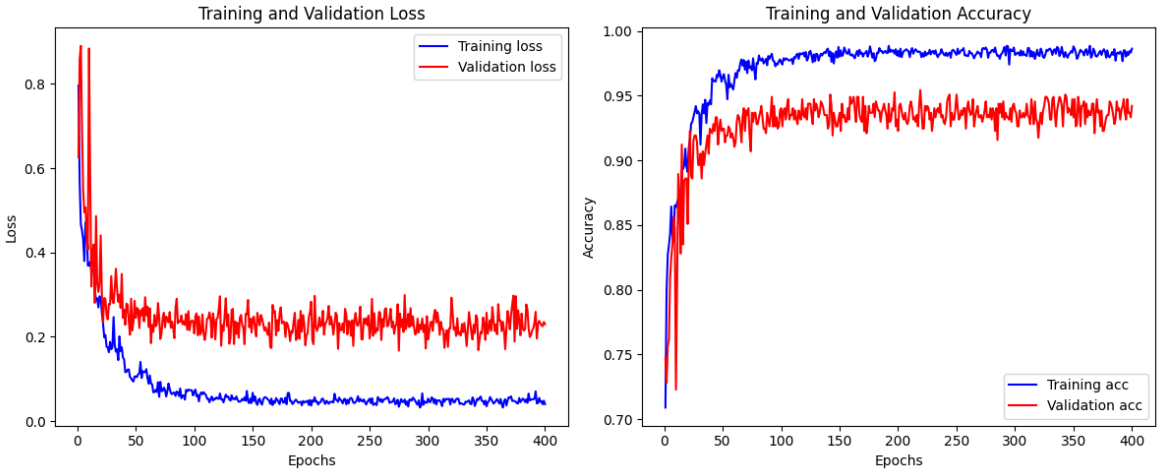

Figure 5: Loss and accuracy curves

The confusion matrix of the two models is shown in Figure 6 and 7, from which we can find that Model 2 has better recognition accuracy than Model 1 for all classes except Daisy. However, both models encountered some issues in distinguishing between rose and tulip.

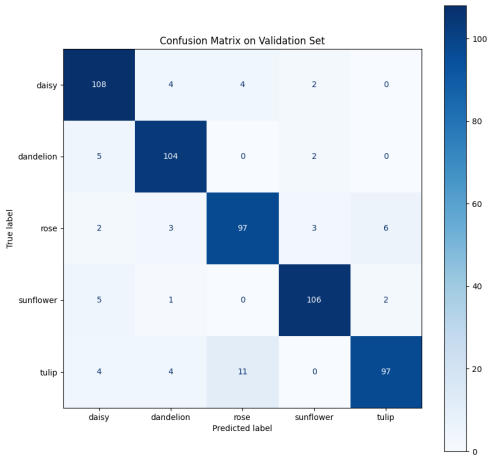

Figure 6: Confusion matrix of Model 1

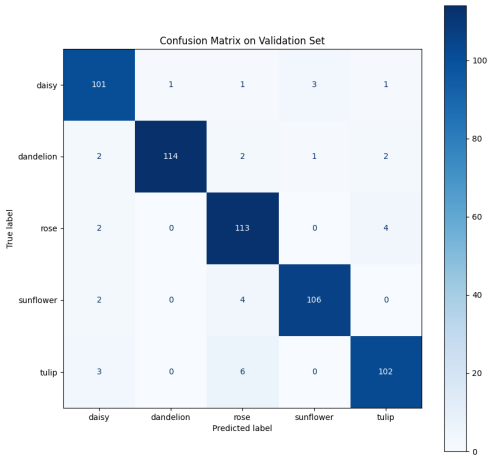

Figure 7: Confusion matrix of Model 2