An image classification model based on ResNet-18 for flower classification.

Junjie Yu 22308235
School of Electronics and Communication Engineering
Sun Yat-sen University, Shenzhen Campus
Yujj53@mail2.sysu.edu.cn

May 2025

Abstract

This report utilizes the resnet18.8xb32.in1k model to perform flower classification on the given dataset. In addition, a simple PyTorch-based training script is implemented to achieve the same classification task. The performance and efficiency of both approaches are compared, and it is ultimately found that the PyTorch script is simpler and yields better results.

Keywords:Neural Network and Machine Learning,Image Recognition

1 Introduction

Based on the two main requirements of the assignment, this report is divided into four sections: Introduction, Task 1, Task 2, and Conclusion. In the sections for Exercise 1 and Exercise 2, the challenges encountered during the process and the corresponding solutions will be discussed in detail. As a beginner in neural networks, completing this assignment was by no means easy. However, through this process, I have gained a deeper understanding of how neural networks work and developed a more profound insight into their underlying principles. In the following sections, I will briefly present the outcomes of my work, the problems I faced, and how I addressed them.

2 Fine-tuning the Classification Model with MMClassification

1) Based on the open-source code available on GitHub, it seems quite straightforward to obtain a training model. Once the environment is properly set up,

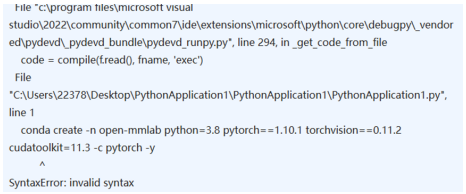

Figure 1: Error1

along with the necessary parameters and training dataset, we can start training our model. While that may sound simple in a single sentence, the actual hands-on experience turned out to be quite a headache.

This was my first time independently working on a project using VS Code, and setting up the environment became my first major challenge. Fortunately, the author of the code provided an install.md file, which served as a helpful guide—like finding stepping stones while crossing a river—for a beginner just starting out with neural networks.

2) The difficulties I encountered fell into several categories. The first type: what kind of environment should the installation commands be executed in?

After installing Miniconda, I knew I had to activate the conda environment, but questions quickly piled up: Should I run git clone inside the conda environment? Where should the pip install commands be executed?

These seemingly basic questions left me confused. I tried repeatedly, faced errors again and again, consulted DeepSeek, searched for tutorials on CSDN. While not particularly difficult in terms of technical complexity, the process made me feel lost—like I didn't know what I was doing—because there was no clear guidance to follow.

3)Figure 1 illustrates one of the problems I faced while configuring the development environment. After successfully setting up the environment, I proceeded to the first step: dataset classification. Here, I used a simple image classification script to organize the dataset as required. Once the data was properly categorized, I began training the model.

4) During the model training process, I used both resnet18.8xb16.cifar10 and resnet18.8xb32.in1k. The former, after 20 training sessions—each with 3,000 iterations and lasting about 3 hours—still struggled to achieve an accuracy of 0.90. Therefore, I switched to the IN1K variant. After 36 training sessions with 72 iterations each, the accuracy reached 0.89, as shown in Figure 2. The results were clearly better, and thus Exercise 1 was completed.

5) Reflecting on the difficulties in Exercise 1, I found the most challenging part was how to properly call train.py along with the configuration file. Since we need to modify paths in the config file, it requires us to understand the structure and purpose of the code.

The second major challenge was environment setup and writing the dataset classification script. Fortunately, thanks to the existing tutorials available online, it was relatively easy to grasp the basic approach.

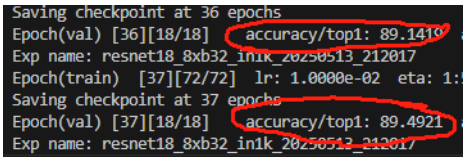

Figure 2: Accuracy

3 Completing the classification model training script

1) This report presents the implementation of a flower image classification task using PyTorch. The goal was to classify images into five categories: daisy, dandelion, rose, sunflower, and tulip. A ResNet18 model pre-trained on ImageNet was fine-tuned for this specific dataset. Data preprocessing, training, evaluation, and model saving were performed in a complete pipeline.

2) In my opinion, Exercise 2 is relatively easier compared to Exercise 1. This task mainly required us to write code, and dealing with pure coding problems is much simpler than managing file interactions and configurations.

The main challenge in this part was understanding the context of the code and figuring out how to implement the required functionality. This also involved searching online for various resources to find a suitable function or code snippet that could fulfill the task.

After trying out multiple options, I was eventually able to identify a function that met the requirements and achieved an accuracy of over 0.9.

3) During the implementation of Exercise 2, several challenges were encountered:

- **UnicodeDecodeError when running the script**: This occurred due to non-UTF-8 characters in the script file. The solution was to reopen the script in VS Code and save it with UTF-8 encoding.
- **ModuleNotFoundError: No module named 'torch'**: This indicated that the script was being run outside the intended Conda environment. Switching to the openmmlab environment using conda activate openmmlab resolved the issue.
- **RuntimeError on multiprocessing**: DataLoader with num_workers greater than 0 caused an error on Windows. Wrapping training code in if name == 'main': prevented the error.
- **Deprecation warning about pretrained argument**: Using pretrained=True raised a deprecation warning. It was resolved by switching to weights=ResNet18.
- **Multiple repeated outputs of model structure**: A misplaced print(model.fc) was printing repeatedly and was removed for cleaner output.

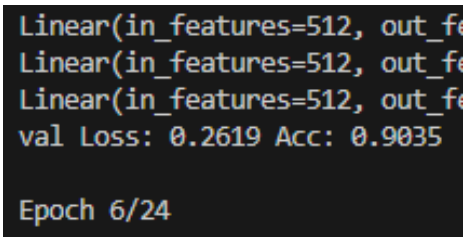

Figure 3: Accuracy 2

These issues helped reinforce good development practices such as environment management, proper multiprocessing setup, and awareness of library API changes.

4) As shown in Figure 3, this model outperforms the one used in Exercise 1.

4 Conclusion

This assignment, through two sub-tasks, comprehensively strengthened my ability to apply deep learning frameworks to image classification tasks.

In the first task, I used the MMClassification toolkit to quickly build and fine-tune a pre-trained model. This helped me understand the standardized workflow involving configuration files, data loaders, and training schedulers.

In the second task, I implemented a complete image classification model from scratch using PyTorch. This allowed me to gain hands-on experience with key components such as model construction, data augmentation, training loops, and model saving.

Throughout the practical process, I encountered various issues, including encoding format errors, multi-processing loading problems, environment switching failures, and API warnings. However, by consulting documentation and performing thorough debugging, I was able to resolve them one by one. These challenges deepened my understanding of Python multi-processing, coding standards, and model weight loading mechanisms. I came to realize that writing runnable code is just the first step—what truly matters is maintaining systematic thinking and strong debugging skills.

Overall, this assignment not only enhanced my understanding of convolutional neural networks but also provided me with a solid foundation in the end-to-end pipeline, from data preprocessing and model training to final deployment. By comparing PyTorch and MMClassification in practice, I am now more confident in selecting appropriate tools for deep learning tasks and independently addressing real-world development challenges.