Cifar10 Classification (HW5)

Student: 張皓雲

Student Id: 310551031

Content

● Part. 1, Coding (100%):

You can see more detailed description in the following github link.

https://github.com/secondlevel/Cifar10-Classification

1. Hardware

```
Operating System: Ubuntu 20.04.3 LTS

CPU: Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz

GPU: NVIDIA GeForce GTX TITAN X
```

2. Requirement

In this part, I use anaconda and pip to build the execution environment.
In addition, the following two options can be used to build an execution environment.

- First Option

```
conda env create -f environment.yml
```

- Second Option

```
conda create --name cifar python=3.8
conda activate cifar
conda install pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch
conda install matplotlib pandas scikit-learn -y
pip install tqdm
```

3. Directory Tree

In this homework, you can put the folder on the specified path according to the pattern in the following directory tree for training and testing.

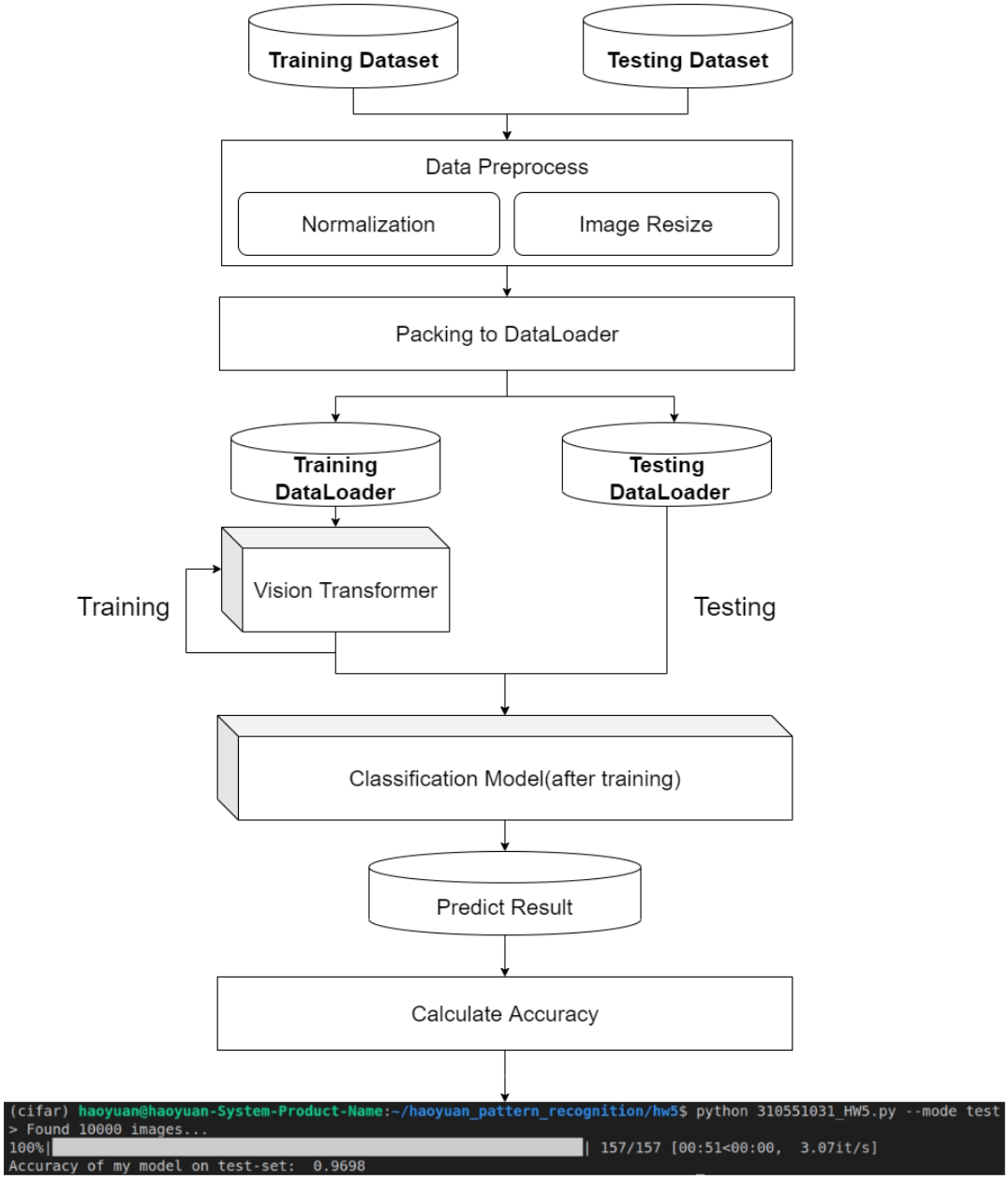
The model weight can be download in the following link, please put it under the checkpoint directory.

The data can be download in the following link, please put it in the under the repository according to the following description.

https://drive.google.com/drive/folders/1Boe0EZT1cyV6MxqqTFk1mufsGbThs4BG?usp=sharing

```
├─ 310551031_HW5.py
├─ environment.yml
├─ history_csv
│  └─ BEST_VIT_CIFAR.csv
├─ checkpoint
│  └─ BEST_VIT_CIFAR.rar
├─ x_train.npy
├─ x_test.npy
├─ y_train.npy
├─ y_test.npy
└─ README.md
```

4. Flow Chart



5. Hyperparameter Setting

```
image_size = 224
number_worker = 4
batch_size = 64
epochs = 10
lr = 2e-5
optimizer = AdamW
loss function = CrossEntropy
```

5.1. Optimizer

In this assignment, I use AdamW to train the model.

```
208     optimizer = optim.AdamW(model.parameters(), lr=lr)
```

5.2. Loss Function

In this assignment, I use CrossEntropy to calculate loss.

```
209     criterion = nn.CrossEntropyLoss()
```

6. Data Preprocess

The Data Preprocess include two parts. The first part is the standardization of pixel value([0, 255] to [0, 1]). The second part is to adjust the image to 224 x 224. Which is implemented by the torchvision.

```
182     # ## Data preprocess
183     train_transform = transforms.Compose([
184         transforms.ToTensor(),  # range [0, 255] -> [0.0,1.0]
185         transforms.Resize((image_size, image_size)),
186     ])
```

7. Data Loader Design

In order to avoid the problem of the cuda out of memory, I create the data loader to process the data. Which is implemented by the torch.utils.

Input: Image Array, Label Array, Data Augmentation method.
Ouput: DataLoader

```python
class CIFARLoader(data.Dataset):
    def __init__(self, image, label, transform=None):

        self.img_name, self.labels = image, label
        self.transform = transform
        print("> Found %d images..." % (len(self.img_name)))

    def __len__(self):
        """'return the size of dataset"""
        return len(self.img_name)

    def __getitem__(self, index):
        """something you should implement here"""

        self.img = self.img_name[index]
        self.label = self.labels[index]

        if self.transform:
            self.img = self.transform(self.img)

        return self.img, self.label
```

8. Model Architecture

In this homework, I used the Vision Transformer pretrained model to classify images.

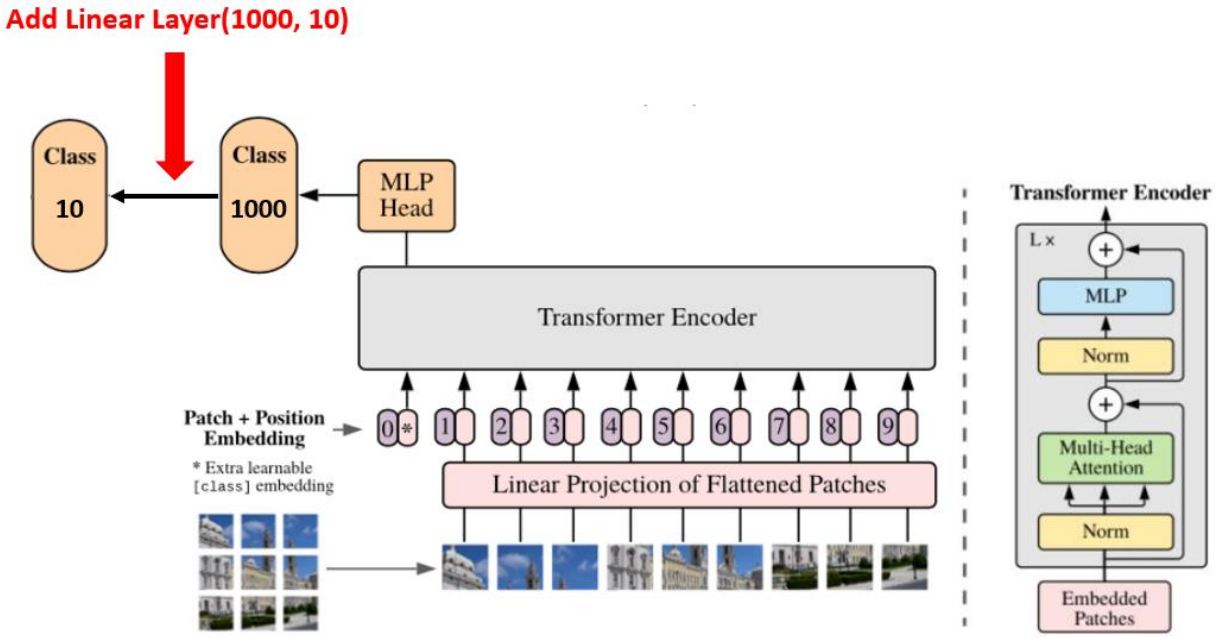
In addition, I added the linear layer to the Vision Transformer(VIT), **all the weight of the VIT is unfreeze**. In the Vision Transformer, it has the transformer encoder, which is composed of Layer Normalization layer, Multi-Head attention, and Multi-layer perceptron. Briefly speaking, Vision Transformer treated the image task to be a language task.

The Architecture of the classification model is as follows.

```python
class VIT(nn.Module):
    def __init__(self, pretrained=True):
        super(VIT, self).__init__()
        self.model = models.vit_b_32(pretrained=pretrained)
        self.classify = nn.Linear(1000, num_classes)

    def forward(self, x):
        x = self.model(x)
        x = self.classify(x)
        return x
```

```
199     model = VIT()
200     print(model)
201
202     for name, child in model.named_children():
203         for param in child.parameters():
204             param.requires_grad = True
205
206     model.cuda(0)
```



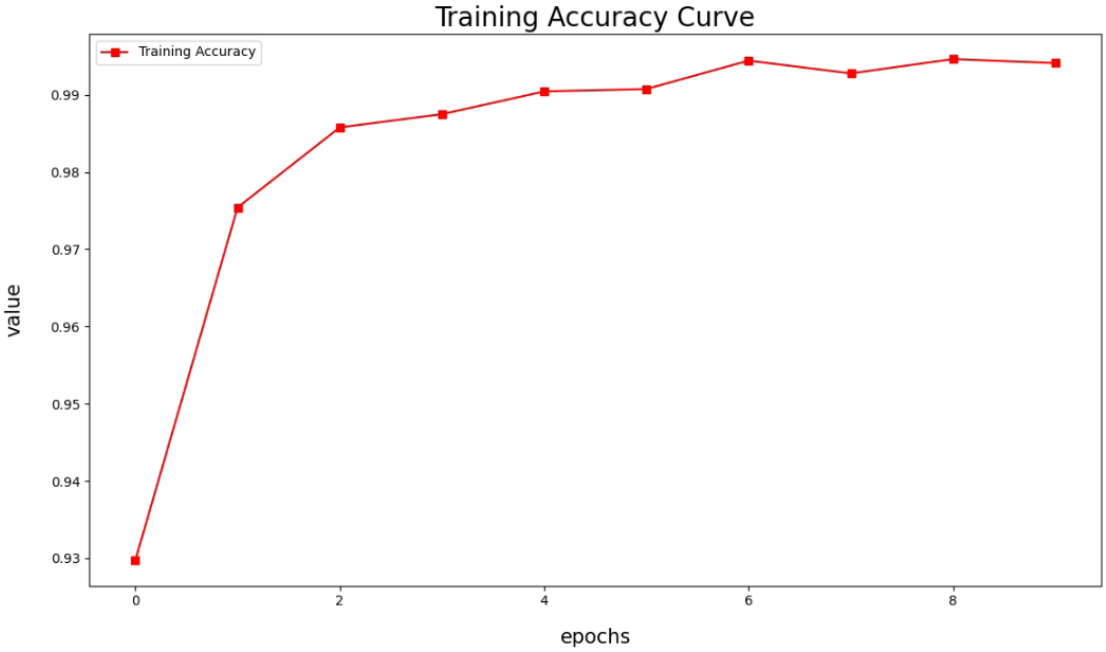
9. Training

You can switch to the training mode with the following instruction, and then you can start training the classification model.

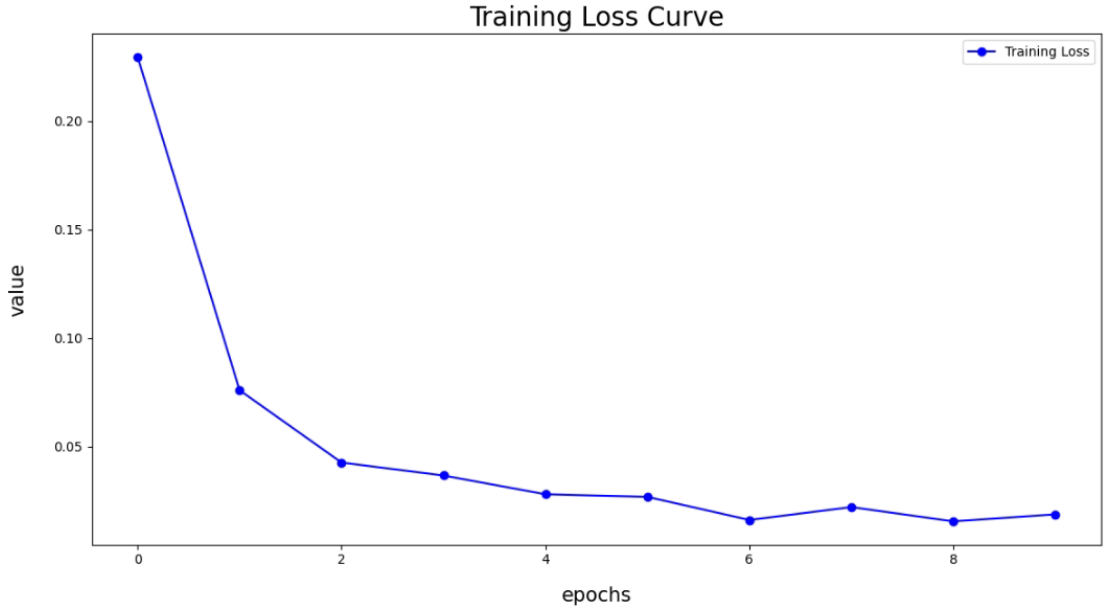
```
python 310551031_HW5.py --mode train
```

The best model weight during training will be stored at **checkpoint directory**, and the training history will in the **history_csv directory**.

The training accuracy history is as following.



The training Loss history is as following.



10. Testing

You can switch to the testing mode with the following instruction, and then you can evaluate the classification result.

Best Model Weight name: BEST_VIT_CIFAR.rar (Which is in the checkpoint directory)

```
python 310551031_HW5.py --mode test
```

```
(cifar) haoyuan@haoyuan-System-Product-Name:~/haoyuan_pattern_recognition/hw5$ python 310551031_HW5.py --mode test
> Found 10000 images...
100%|████████████████████████████████████████| 157/157 [00:51<00:00,  3.07it/s]
Accuracy of my model on test-set:  0.9698
```

```
283     model = VIT()
284     model.cuda(0)
285
286     model.load_state_dict(torch.load(test_filepath))
287     y_pred = predict(test_loader, model, device)
288     y_pred = np.array(y_pred)
289
290     assert y_pred.shape == (10000,)
291
292     y_test = np.load("y_test.npy")
293     print("Accuracy of my model on test-set: ", accuracy_score(y_test, y_pred))
```