学习汇报

焦强

论文

李宏毅的作业三

论文

有创意的点是：
1，提出了一种从FPGA端向片外DRAM发起数据访问的一种系统架构。在该架构中FPGA可以直接对片外DRAM进行访问，不需要处理器进行干预。
2，一个可行的量化方法，确定量化精度的可能范围，再从中选取最优量化精度的方法，试图降低数据量化花费的时间成本。

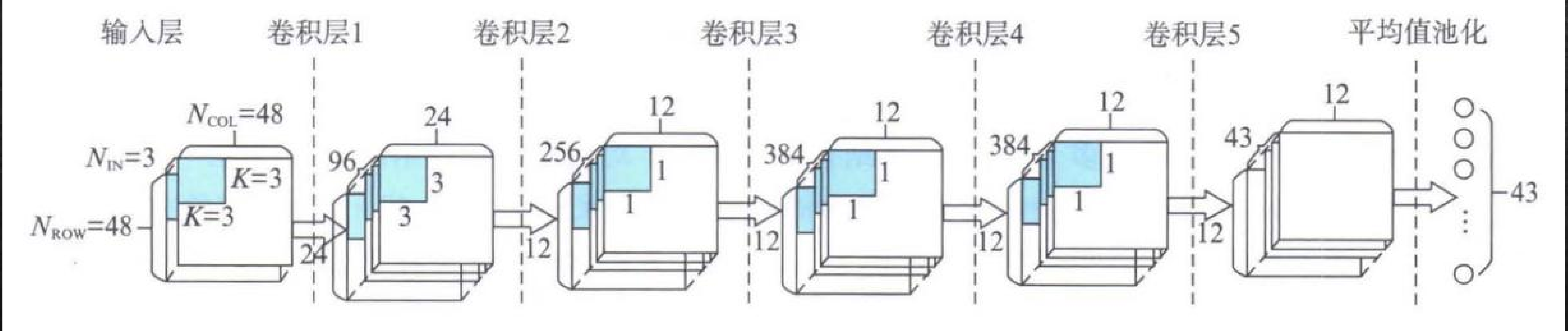


图 1 卷积神经网络结构

数据传输的优化

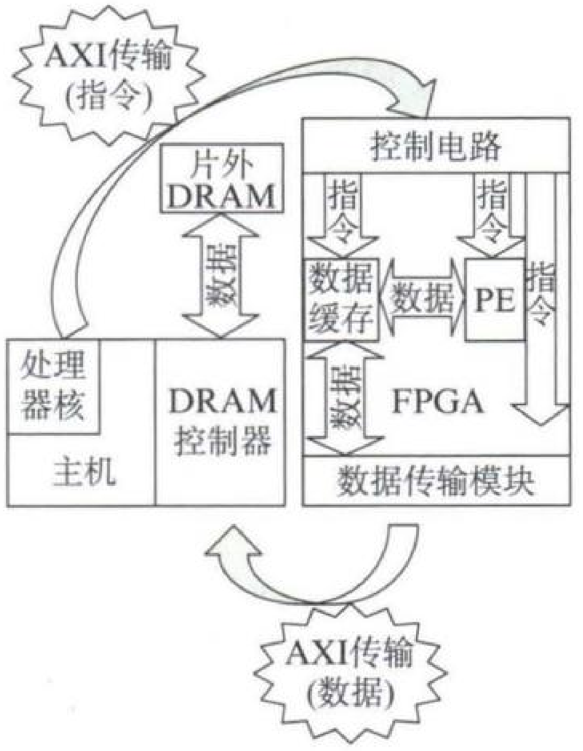


图 2　系统架构

一种从FPGA端发起数据传输的系统架构。

在识别开始时，由处理器对FPGA电路进行初始配置，FPGA收到开始信号后，片上缓存和PE分别在控制电路的支配下开始进行数据缓存和处理．在识别过程中控制电路可以根据当前系统的状态直接向数据传输模块发送指令，进而发起对片外DRAM的数据传输，数据请求成功后直接通过数据传输模块加载到数据缓存中，整个传输过程不需要处理器参与。

在整个识别过程中处理器仅仅在识别开始和识别结束时与FPGA发生通信，显著降低识别过程中处理器与FPGA通信所需的时间。

网络压缩

$2^{W-P-1} > \max(|D_{\max}| \times 2^P, |D_{\min}| \times 2^P)$

P为量化精度；w为量化位宽；D表示量化前的浮点数据。

浮点数---整数 $q = round(\frac{r}{S} + Z)$

这里，S和Z定义：

$$S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}}$$

$$Z = round(q_{max} - \frac{r_{max}}{S})$$

$S_3(q_3^{(i,j)} - Z_3) = \sum S_1(q_1^{(i,j)} - Z_1)S_2(q_2^{(i,j)} - Z_2)$ --- $q_3^{(i,j)} = M\sum(q_1^{(i,j)} - Z_1)(q_2^{(i,j)} - Z_2) + Z_3 = MP + Z_3$

这里，M定义为：$M = \frac{S_1 S_2}{S_3}$

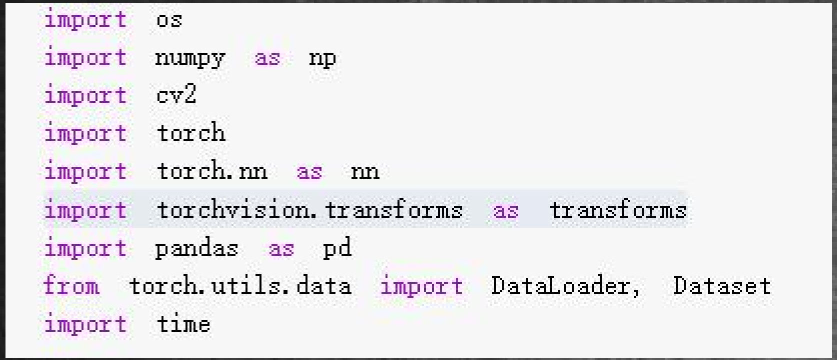
把M转换为整数：$M = 2^{-n} M_0$

最后，把整数反量化：$r = S(q - Z)$

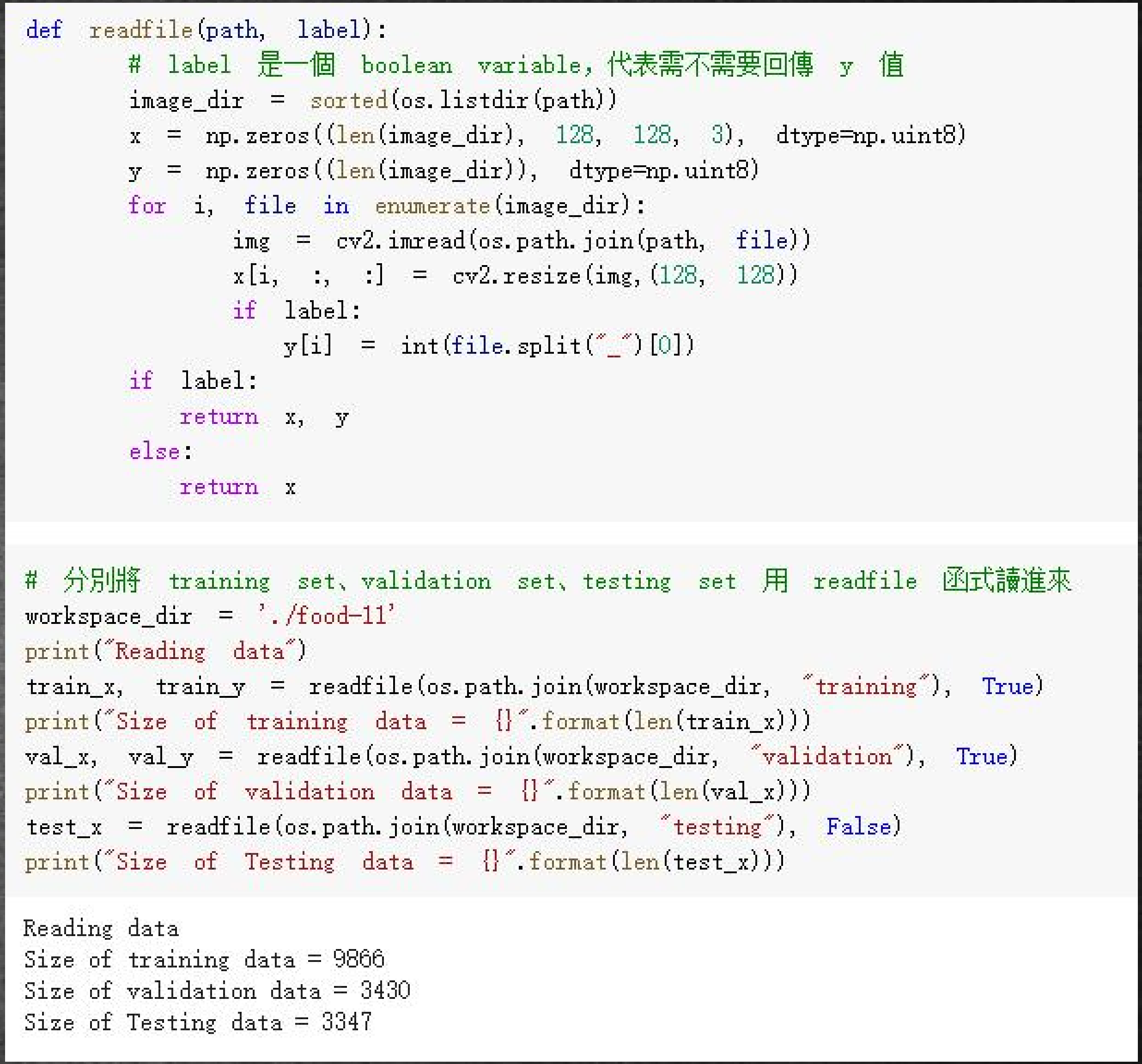
步骤：
1，提前计算出所有的S和Z；
2，提前计算出n和M0；
3，M0乘上剩余部分获取一个整数。
4，将得到的整数，进行n次右移，获得最终整数；
5，再把得到的整数反量化。

作业三

作业三：食物的识别



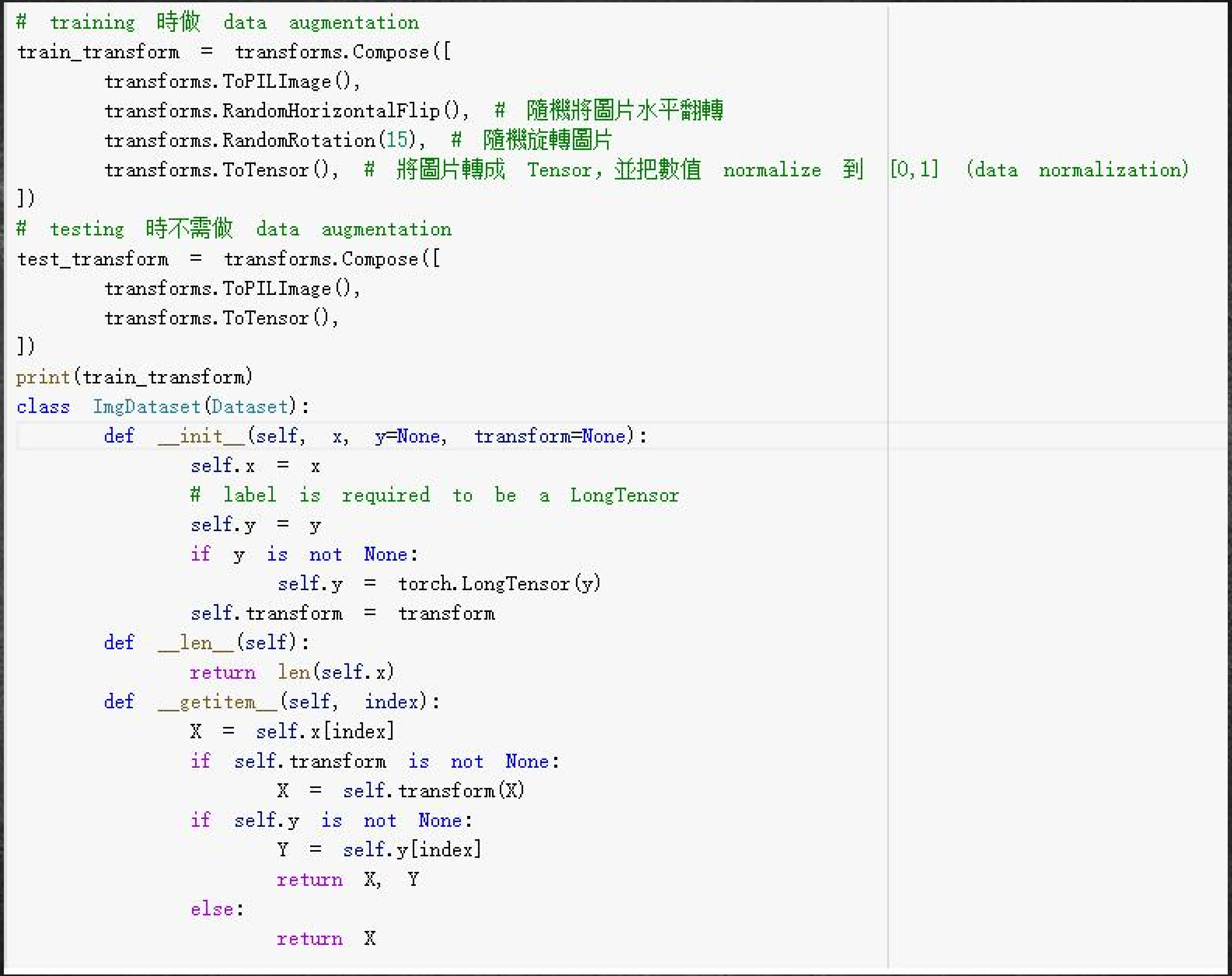
```
import os
import numpy as np
import cv2
import torch
import torch.nn as nn
import torchvision.transforms as transforms
import pandas as pd
from torch.utils.data import DataLoader, Dataset
import time
```



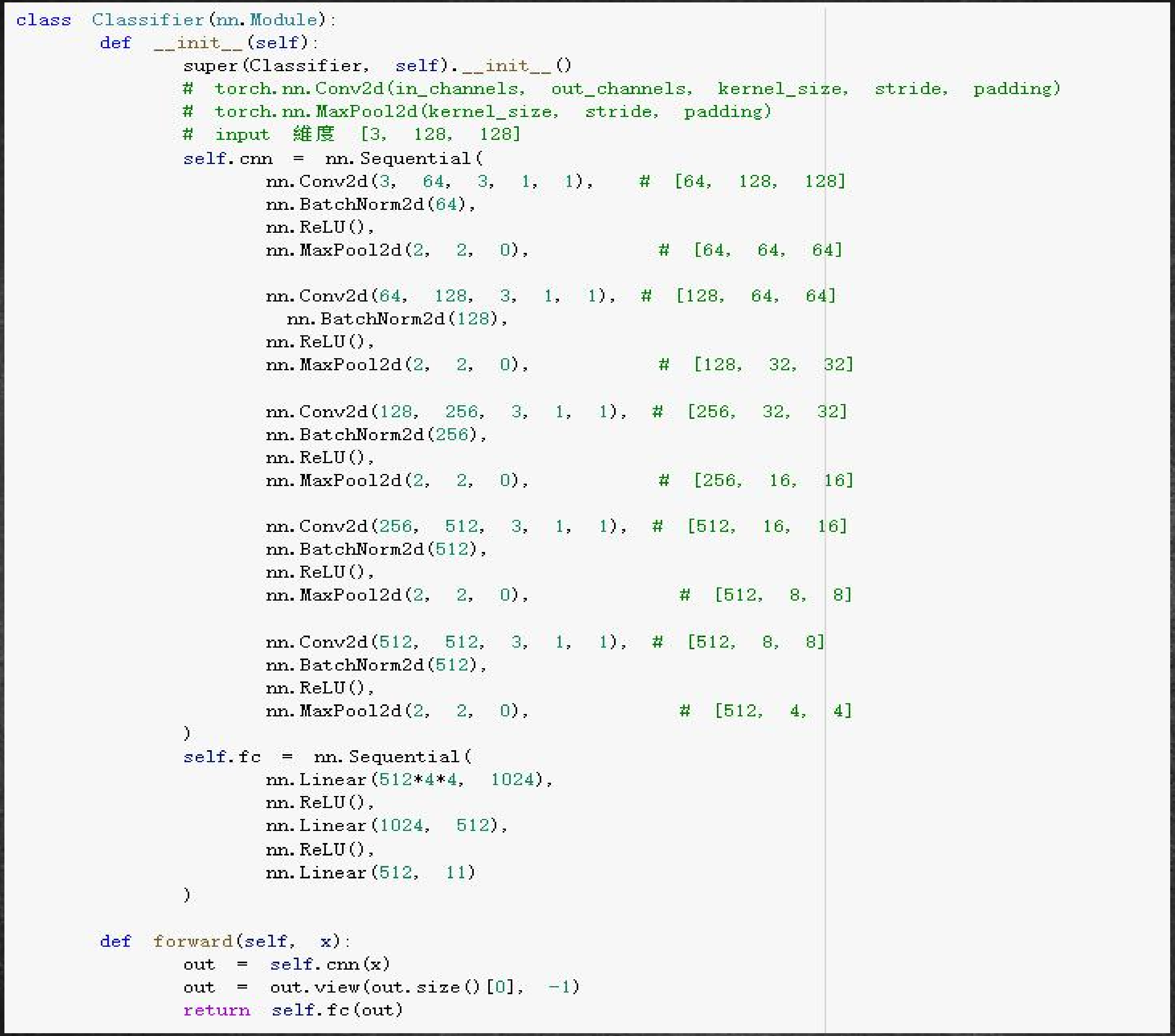
```
def readfile(path, label):
    # label 是一個 boolean variable，代表需不需要回傳 y 值
    image_dir = sorted(os.listdir(path))
    x = np.zeros((len(image_dir), 128, 128, 3), dtype=np.uint8)
    y = np.zeros((len(image_dir)), dtype=np.uint8)
    for i, file in enumerate(image_dir):
        img = cv2.imread(os.path.join(path, file))
        x[i, :, :] = cv2.resize(img,(128, 128))
        if label:
          y[i] = int(file.split("_")[0])
    if label:
      return x, y
    else:
      return x
```

```
# 分別將 training set、validation set、testing set 用 readfile 函式讀進來
workspace_dir = './food-11'
print("Reading data")
train_x, train_y = readfile(os.path.join(workspace_dir, "training"), True)
print("Size of training data = {}".format(len(train_x)))
val_x, val_y = readfile(os.path.join(workspace_dir, "validation"), True)
print("Size of validation data = {}".format(len(val_x)))
test_x = readfile(os.path.join(workspace_dir, "testing"), False)
print("Size of Testing data = {}".format(len(test_x)))
```

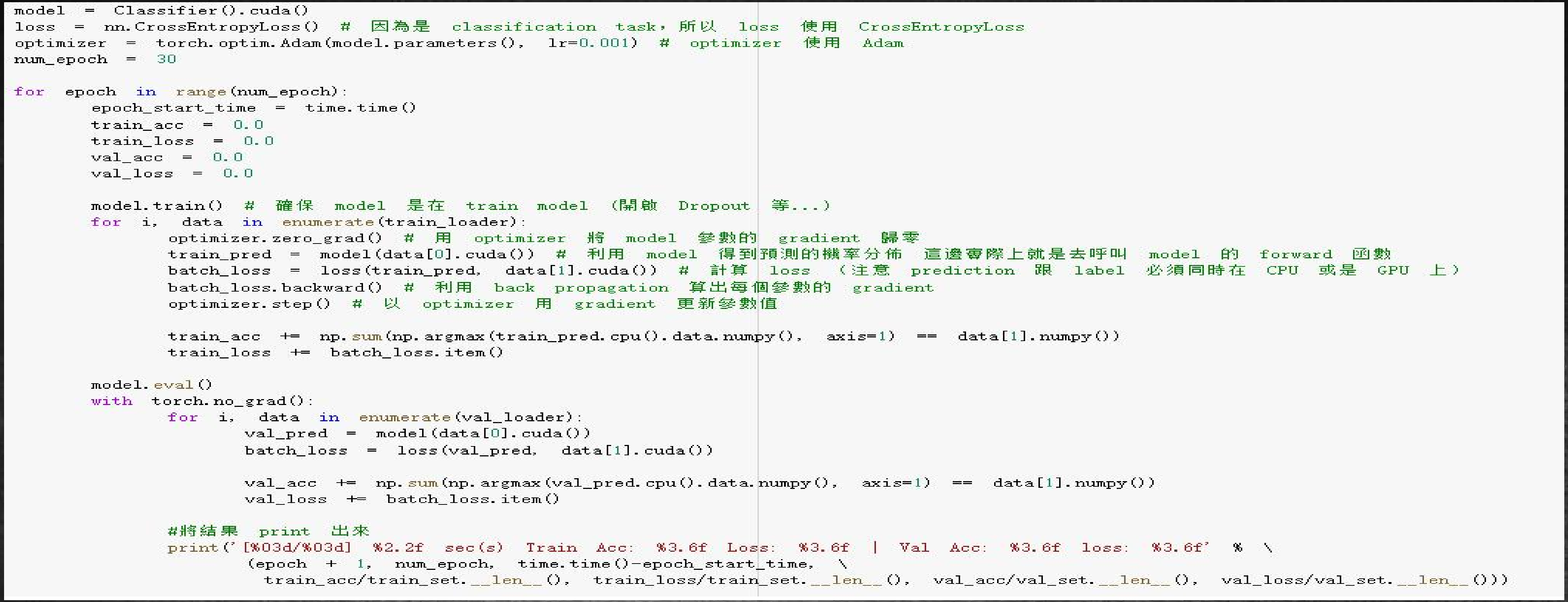
```
Reading data
Size of training data = 9866
Size of validation data = 3430
Size of Testing data = 3347
```



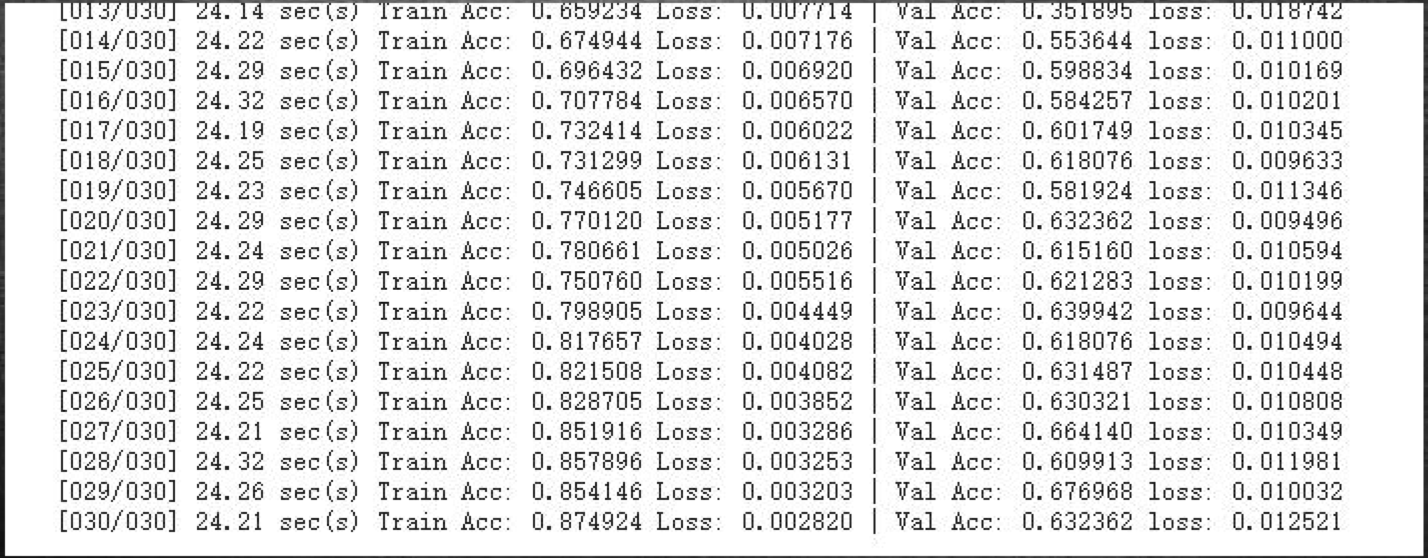
```python
# training 時做 data augmentation
train_transform = transforms.Compose([
        transforms.ToPILImage(),
        transforms.RandomHorizontalFlip(), # 隨機將圖片水平翻轉
        transforms.RandomRotation(15), # 隨機旋轉圖片
        transforms.ToTensor(), # 將圖片轉成 Tensor，並把數值 normalize 到 [0,1] (data normalization)
])
# testing 時不需做 data augmentation
test_transform = transforms.Compose([
        transforms.ToPILImage(),
        transforms.ToTensor(),
])
print(train_transform)
class ImgDataset(Dataset):
        def __init__(self, x, y=None, transform=None):
                self.x = x
                # label is required to be a LongTensor
                self.y = y
                if y is not None:
                        self.y = torch.LongTensor(y)
                self.transform = transform
        def __len__(self):
                return len(self.x)
        def __getitem__(self, index):
                X = self.x[index]
                if self.transform is not None:
                        X = self.transform(X)
                if self.y is not None:
                        Y = self.y[index]
                        return X, Y
                else:
                        return X
```



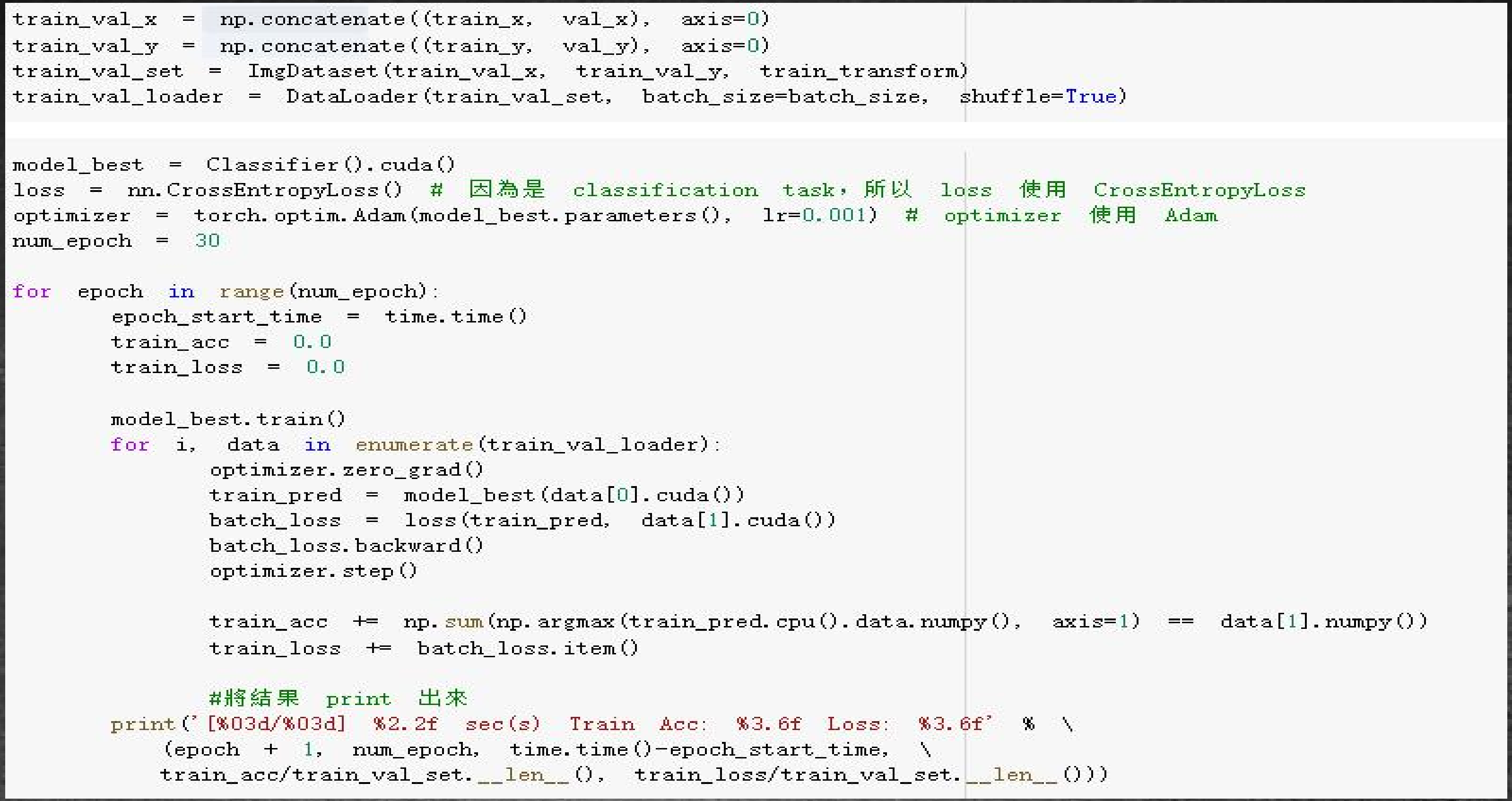
```python
class Classifier(nn.Module):
    def __init__(self):
        super(Classifier, self).__init__()
        # torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding)
        # torch.nn.MaxPool2d(kernel_size, stride, padding)
        # input 維度 [3, 128, 128]
        self.cnn = nn.Sequential(
            nn.Conv2d(3, 64, 3, 1, 1),    # [64, 128, 128]
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),        # [64, 64, 64]

            nn.Conv2d(64, 128, 3, 1, 1),  # [128, 64, 64]
              nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),        # [128, 32, 32]

            nn.Conv2d(128, 256, 3, 1, 1), # [256, 32, 32]
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),        # [256, 16, 16]

            nn.Conv2d(256, 512, 3, 1, 1), # [512, 16, 16]
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),          # [512, 8, 8]

            nn.Conv2d(512, 512, 3, 1, 1), # [512, 8, 8]
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),          # [512, 4, 4]
        )
        self.fc = nn.Sequential(
            nn.Linear(512*4*4, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 11)
        )

    def forward(self, x):
        out = self.cnn(x)
        out = out.view(out.size()[0], -1)
        return self.fc(out)
```



```python
model = Classifier().cuda()
loss = nn.CrossEntropyLoss() # 因為是 classification task，所以 loss 使用 CrossEntropyLoss
optimizer = torch.optim.Adam(model.parameters(), lr=0.001) # optimizer 使用 Adam
num_epoch = 30

for epoch in range(num_epoch):
    epoch_start_time = time.time()
    train_acc = 0.0
    train_loss = 0.0
    val_acc = 0.0
    val_loss = 0.0

    model.train() # 確保 model 是在 train model (開啟 Dropout 等...)
    for i, data in enumerate(train_loader):
        optimizer.zero_grad() # 用 optimizer 將 model 參數的 gradient 歸零
        train_pred = model(data[0].cuda()) # 利用 model 得到預測的機率分佈 這邊實際上就是去呼叫 model 的 forward 函數
        batch_loss = loss(train_pred, data[1].cuda()) # 計算 loss （注意 prediction 跟 label 必須同時在 CPU 或是 GPU 上）
        batch_loss.backward() # 利用 back propagation 算出每個參數的 gradient
        optimizer.step() # 以 optimizer 用 gradient 更新參數值

        train_acc += np.sum(np.argmax(train_pred.cpu().data.numpy(), axis=1) == data[1].numpy())
        train_loss += batch_loss.item()

    model.eval()
    with torch.no_grad():
        for i, data in enumerate(val_loader):
            val_pred = model(data[0].cuda())
            batch_loss = loss(val_pred, data[1].cuda())

            val_acc += np.sum(np.argmax(val_pred.cpu().data.numpy(), axis=1) == data[1].numpy())
            val_loss += batch_loss.item()

        #將結果 print 出來
        print('[%03d/%03d] %2.2f sec(s) Train Acc: %3.6f Loss: %3.6f | Val Acc: %3.6f loss: %3.6f' % \
            (epoch + 1, num_epoch, time.time()-epoch_start_time, \
             train_acc/train_set.__len__(), train_loss/train_set.__len__(), val_acc/val_set.__len__(), val_loss/val_set.__len__()))
```

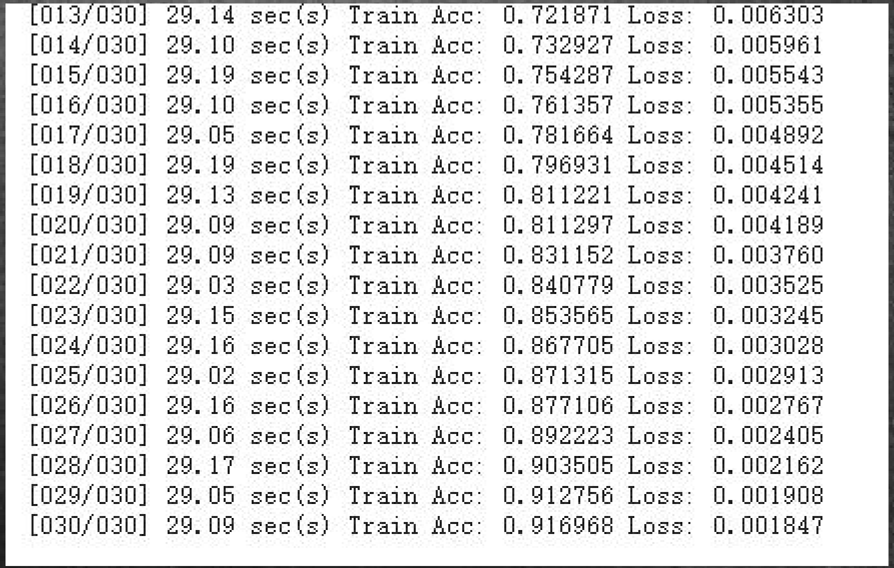


```
[013/030] 24.14 sec(s) Train Acc: 0.659234 Loss: 0.007714 | Val Acc: 0.351895 loss: 0.018742
[014/030] 24.22 sec(s) Train Acc: 0.674944 Loss: 0.007176 | Val Acc: 0.553644 loss: 0.011000
[015/030] 24.29 sec(s) Train Acc: 0.696432 Loss: 0.006920 | Val Acc: 0.598834 loss: 0.010169
[016/030] 24.32 sec(s) Train Acc: 0.707784 Loss: 0.006570 | Val Acc: 0.584257 loss: 0.010201
[017/030] 24.19 sec(s) Train Acc: 0.732414 Loss: 0.006022 | Val Acc: 0.601749 loss: 0.010345
[018/030] 24.25 sec(s) Train Acc: 0.731299 Loss: 0.006131 | Val Acc: 0.618076 loss: 0.009633
[019/030] 24.23 sec(s) Train Acc: 0.746605 Loss: 0.005670 | Val Acc: 0.581924 loss: 0.011346
[020/030] 24.29 sec(s) Train Acc: 0.770120 Loss: 0.005177 | Val Acc: 0.632362 loss: 0.009496
[021/030] 24.24 sec(s) Train Acc: 0.780661 Loss: 0.005026 | Val Acc: 0.615160 loss: 0.010594
[022/030] 24.29 sec(s) Train Acc: 0.750760 Loss: 0.005516 | Val Acc: 0.621283 loss: 0.010199
[023/030] 24.22 sec(s) Train Acc: 0.798905 Loss: 0.004449 | Val Acc: 0.639942 loss: 0.009644
[024/030] 24.24 sec(s) Train Acc: 0.817657 Loss: 0.004028 | Val Acc: 0.618076 loss: 0.010494
[025/030] 24.22 sec(s) Train Acc: 0.821508 Loss: 0.004082 | Val Acc: 0.631487 loss: 0.010448
[026/030] 24.25 sec(s) Train Acc: 0.828705 Loss: 0.003852 | Val Acc: 0.630321 loss: 0.010808
[027/030] 24.21 sec(s) Train Acc: 0.851916 Loss: 0.003286 | Val Acc: 0.664140 loss: 0.010349
[028/030] 24.32 sec(s) Train Acc: 0.857896 Loss: 0.003253 | Val Acc: 0.609913 loss: 0.011981
[029/030] 24.26 sec(s) Train Acc: 0.854146 Loss: 0.003203 | Val Acc: 0.676968 loss: 0.010032
[030/030] 24.21 sec(s) Train Acc: 0.874924 Loss: 0.002820 | Val Acc: 0.632362 loss: 0.012521
```



```python
train_val_x = np.concatenate((train_x, val_x), axis=0)
train_val_y = np.concatenate((train_y, val_y), axis=0)
train_val_set = ImgDataset(train_val_x, train_val_y, train_transform)
train_val_loader = DataLoader(train_val_set, batch_size=batch_size, shuffle=True)
```

```python
model_best = Classifier().cuda()
loss = nn.CrossEntropyLoss() # 因為是 classification task，所以 loss 使用 CrossEntropyLoss
optimizer = torch.optim.Adam(model_best.parameters(), lr=0.001) # optimizer 使用 Adam
num_epoch = 30

for epoch in range(num_epoch):
    epoch_start_time = time.time()
    train_acc = 0.0
    train_loss = 0.0

    model_best.train()
    for i, data in enumerate(train_val_loader):
        optimizer.zero_grad()
        train_pred = model_best(data[0].cuda())
        batch_loss = loss(train_pred, data[1].cuda())
        batch_loss.backward()
        optimizer.step()

        train_acc += np.sum(np.argmax(train_pred.cpu().data.numpy(), axis=1) == data[1].numpy())
        train_loss += batch_loss.item()

        #將結果 print 出來
    print('[%03d/%03d] %2.2f sec(s) Train Acc: %3.6f Loss: %3.6f' % \
        (epoch + 1, num_epoch, time.time()-epoch_start_time, \
        train_acc/train_val_set.__len__(), train_loss/train_val_set.__len__()))
```



```
[013/030] 29.14 sec(s) Train Acc: 0.721871 Loss: 0.006303
[014/030] 29.10 sec(s) Train Acc: 0.732927 Loss: 0.005961
[015/030] 29.19 sec(s) Train Acc: 0.754287 Loss: 0.005543
[016/030] 29.10 sec(s) Train Acc: 0.761357 Loss: 0.005355
[017/030] 29.05 sec(s) Train Acc: 0.781664 Loss: 0.004892
[018/030] 29.19 sec(s) Train Acc: 0.796931 Loss: 0.004514
[019/030] 29.13 sec(s) Train Acc: 0.811221 Loss: 0.004241
[020/030] 29.09 sec(s) Train Acc: 0.811297 Loss: 0.004189
[021/030] 29.09 sec(s) Train Acc: 0.831152 Loss: 0.003760
[022/030] 29.03 sec(s) Train Acc: 0.840779 Loss: 0.003525
[023/030] 29.15 sec(s) Train Acc: 0.853565 Loss: 0.003245
[024/030] 29.16 sec(s) Train Acc: 0.867705 Loss: 0.003028
[025/030] 29.02 sec(s) Train Acc: 0.871315 Loss: 0.002913
[026/030] 29.16 sec(s) Train Acc: 0.877106 Loss: 0.002767
[027/030] 29.06 sec(s) Train Acc: 0.892223 Loss: 0.002405
[028/030] 29.17 sec(s) Train Acc: 0.903505 Loss: 0.002162
[029/030] 29.05 sec(s) Train Acc: 0.912756 Loss: 0.001908
[030/030] 29.09 sec(s) Train Acc: 0.916968 Loss: 0.001847
```

下周：论文，NLP相关知识。UG902实验