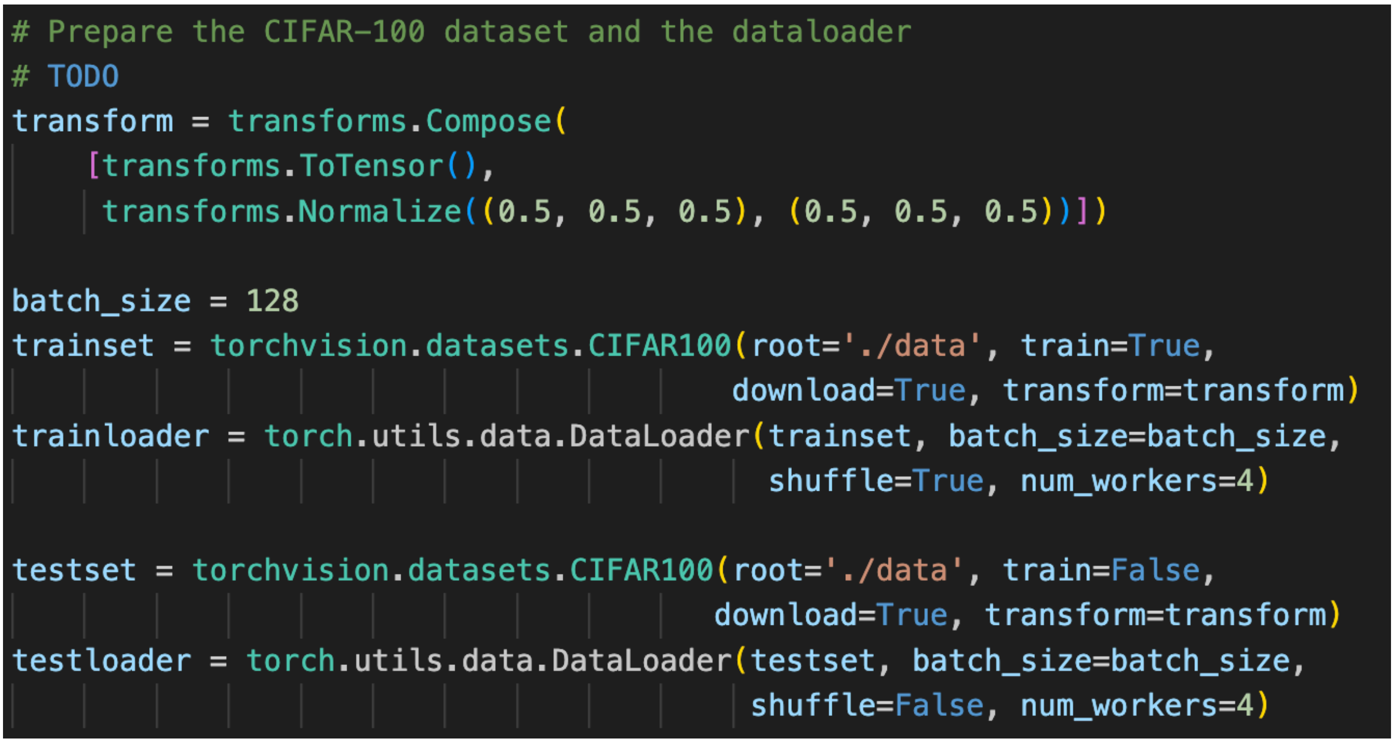
Hw11: Build a Real ResNet

周亦涵 2020012853 未央-水木01

Part1

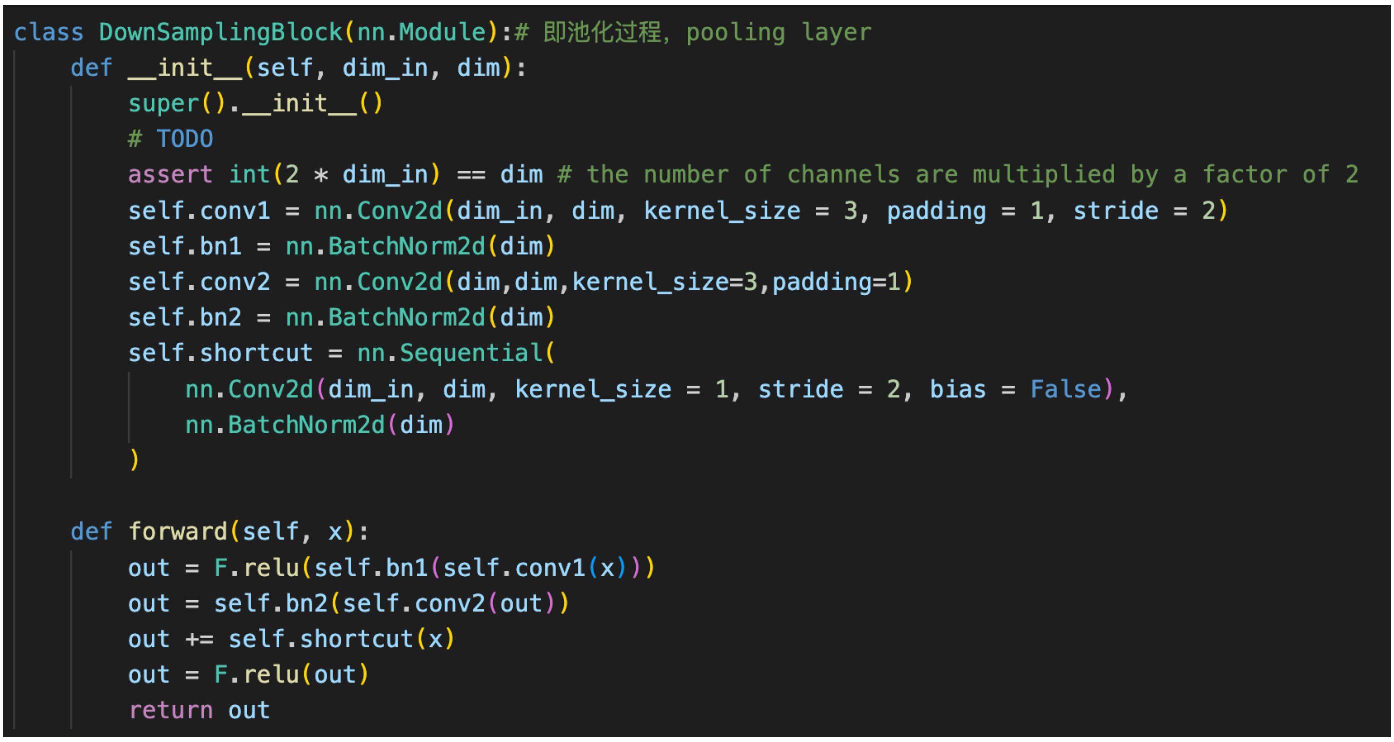
> *5 Screenshots of each part of the code*

1. Prepare the CIFAR-100 dataset and the dataloader



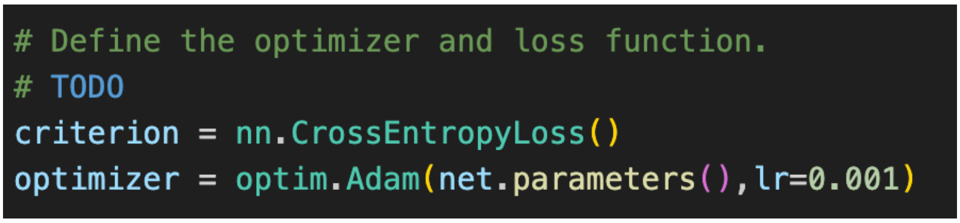
```python
# Prepare the CIFAR-100 dataset and the dataloader
# TODO
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

batch_size = 128
trainset = torchvision.datasets.CIFAR100(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=4)

testset = torchvision.datasets.CIFAR100(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=4)
```

2. The multi-stage ResNet



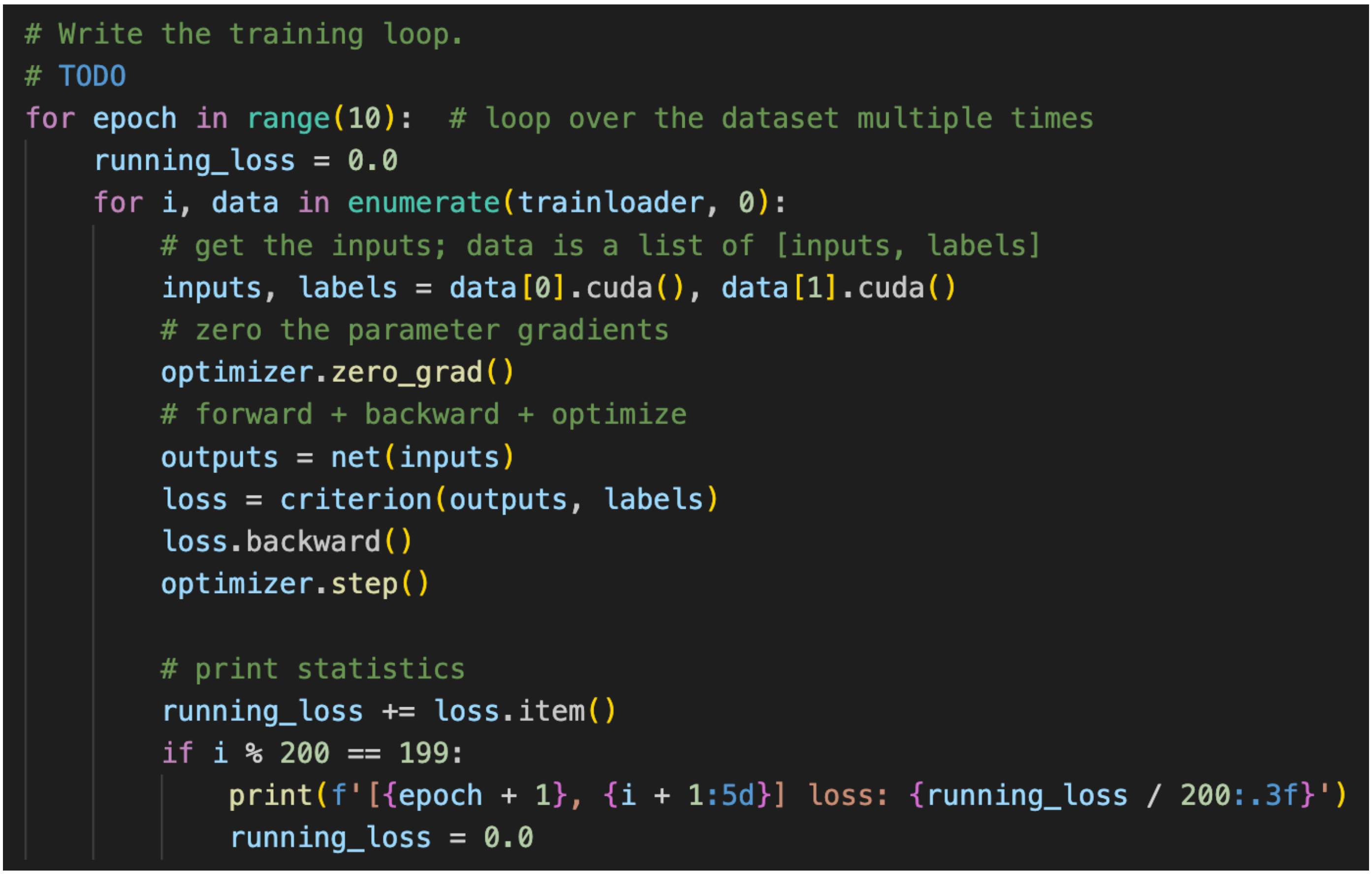
```
class DownSamplingBlock(nn.Module):# 即池化过程，pooling layer
    def __init__(self, dim_in, dim):
        super().__init__()
        # TODO
        assert int(2 * dim_in) == dim # the number of channels are multiplied by a factor of 2
        self.conv1 = nn.Conv2d(dim_in, dim, kernel_size = 3, padding = 1, stride = 2)
        self.bn1 = nn.BatchNorm2d(dim)
        self.conv2 = nn.Conv2d(dim,dim,kernel_size=3,padding=1)
        self.bn2 = nn.BatchNorm2d(dim)
        self.shortcut = nn.Sequential(
            nn.Conv2d(dim_in, dim, kernel_size = 1, stride = 2, bias = False),
            nn.BatchNorm2d(dim)
        )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

3. Define the optimizer and loss function



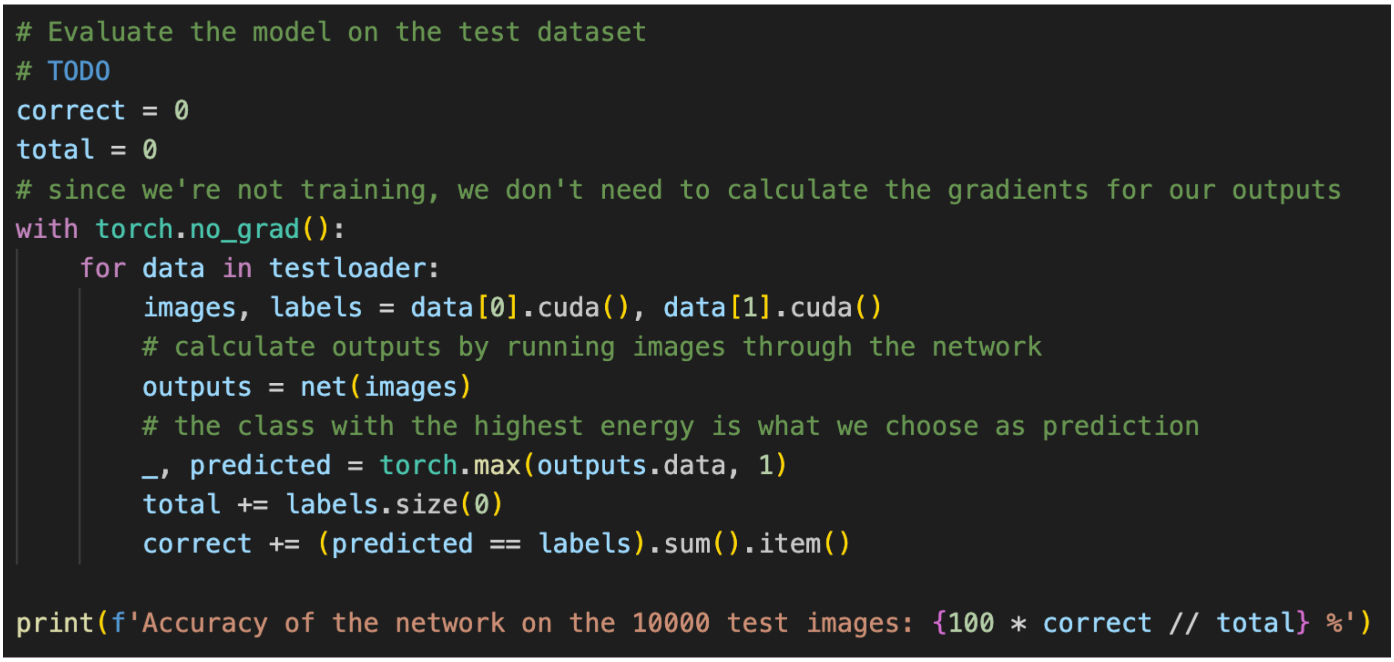
```
# Define the optimizer and loss function.
# TODO
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(),lr=0.001)
```

4. Write the training loop and perform training for 10 epochs.



```
# Write the training loop.
# TODO
for epoch in range(10):  # loop over the dataset multiple times
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data[0].cuda(), data[1].cuda()
        # zero the parameter gradients
        optimizer.zero_grad()
        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 200 == 199:
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 200:.3f}')
            running_loss = 0.0
```

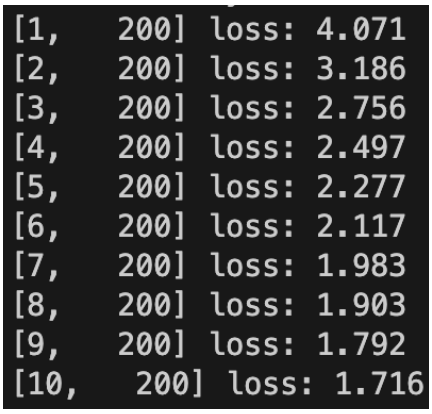
5. Evaluate the model on the test dataset



```
# Evaluate the model on the test dataset
# TODO
correct = 0
total = 0
# since we're not training, we don't need to calculate the gradients for our outputs
with torch.no_grad():
    for data in testloader:
        images, labels = data[0].cuda(), data[1].cuda()
        # calculate outputs by running images through the network
        outputs = net(images)
        # the class with the highest energy is what we choose as prediction
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 * correct // total} %')
```

Part2

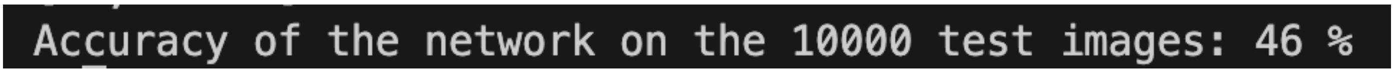
> *The mean training loss of each epoch:*



```
[1,   200] loss: 4.071
[2,   200] loss: 3.186
[3,   200] loss: 2.756
[4,   200] loss: 2.497
[5,   200] loss: 2.277
[6,   200] loss: 2.117
[7,   200] loss: 1.983
[8,   200] loss: 1.903
[9,   200] loss: 1.792
[10,   200] loss: 1.716
```

Part3

> *The accuracy of the network on the 10000 test images (which should be more than 40%):* 46%



```
Accuracy of the network on the 10000 test images: 46 %
```

Part4

The complete code of `main.py` is attached in the folder.