NNI学生项目2020

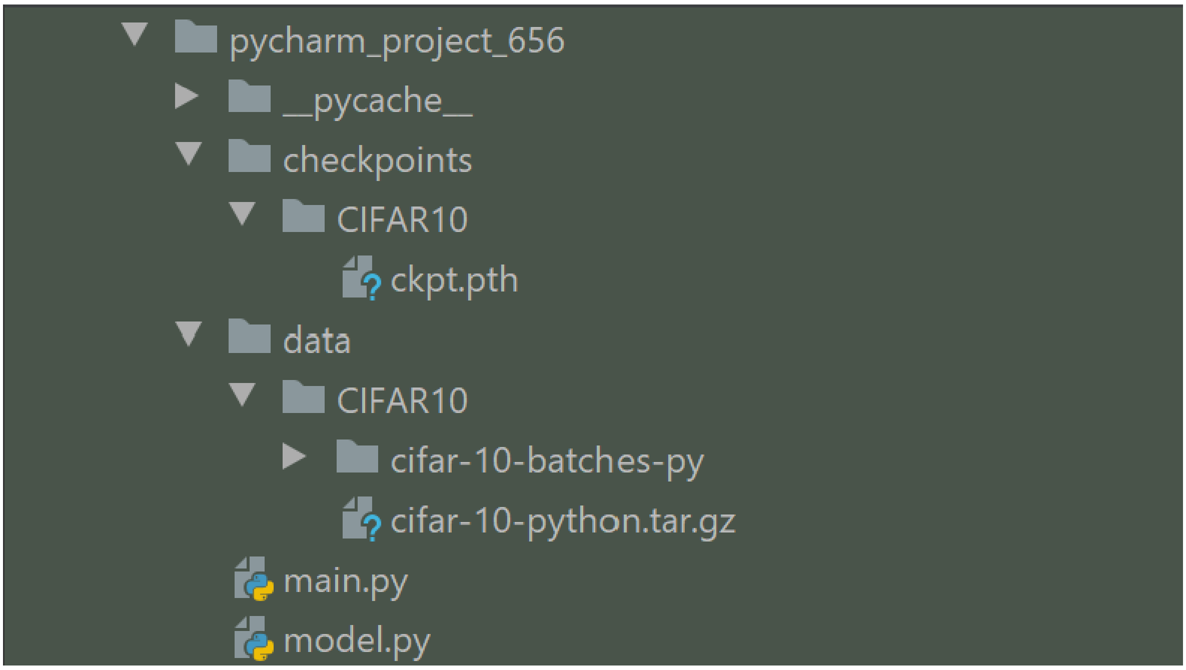
Task 1.2.1

ID: 14 Name: 中科大飙车队

任务描述

- 访问Python机器学习库网站
- 完成初级任务: Training a classifier
- 以CIFAR10为样例，通过图像分类，训练小型神经网络，加深同学对PyTorch Tensor Library和神经网络的理解，帮助初级同学补充机器学习相关的背景知识。
- 提交相关文档，包括配置文件、代码、实现、结果。

代码结构



- checkpoints/ :保存模型参数
- data/CIFAR10/ :数据集所在地
- main.py: 主程序的运行
- model.py: pytorch模型所在地

模型参数

我们使用的模型是类似于VGG架构但比VGG模型参数要少的模型(后文中称为smallVGG). VGG的基本结构是由两个卷积层+一个池化层组成的。我们使用的smallVGG由三个这样的结构组成。具体的结构我们使用torchsummary工具包进行呈现如下：

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [32, 64, 64, 64]           1,792
              ReLU-2           [32, 64, 64, 64]               0
            Conv2d-3           [32, 64, 64, 64]          36,928
```

```
            ReLU-4            [32, 64, 64, 64]               0
         MaxPool2d-5            [32, 64, 32, 32]               0
            Conv2d-6           [32, 128, 32, 32]          73,856
              ReLU-7           [32, 128, 32, 32]               0
            Conv2d-8           [32, 128, 32, 32]         147,584
              ReLU-9           [32, 128, 32, 32]               0
        MaxPool2d-10           [32, 128, 16, 16]               0
           Conv2d-11           [32, 128, 16, 16]         147,584
             ReLU-12           [32, 128, 16, 16]               0
           Conv2d-13           [32, 128, 16, 16]         147,584
             ReLU-14           [32, 128, 16, 16]               0
        MaxPool2d-15             [32, 128, 8, 8]               0
           Linear-16                   [32, 512]       1,049,088
          Sigmoid-17                   [32, 512]               0
           Linear-18                    [32, 10]           5,130
================================================================
Total params: 1,609,546
Trainable params: 1,609,546
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 1.50
Forward/backward pass size (MB): 442.25
Params size (MB): 6.14
Estimated Total Size (MB): 449.89
----------------------------------------------------------------
```

我们选取batch size为32.选取的优化器为Adam优化器，学习率为0.001而其他参数符合默认。我们每20epoches对学习率进行减半。

代码实现

model.py

```
#import *** omitted for simplicity
class smallVGG(nn.Module):
    def __init__(self, NChannels):
        super(smallVGG, self).__init__()
        self.features = []
        self.layerDict = collections.OrderedDict()

        self.conv11 = nn.Conv2d(
            in_channels = NChannels,
            out_channels = 64,
            kernel_size = 3,
            padding = 1
        )
        self.features.append(self.conv11)
        self.layerDict['conv11'] = self.conv11

        self.ReLU11 = nn.ReLU()
        self.features.append(self.ReLU11)
        self.layerDict['ReLU11'] = self.ReLU11

        self.conv12 = nn.Conv2d(
            in_channels = 64,
            out_channels = 64,
            kernel_size = 3,
```

```
            padding = 1
        )
        self.features.append(self.conv12)
        self.layerDict['conv12'] = self.conv12

        self.ReLU12 = nn.ReLU()
        self.features.append(self.ReLU12)
        self.layerDict['ReLU12'] = self.ReLU12

        self.pool1 = nn.MaxPool2d(2,2)
        self.features.append(self.pool1)
        self.layerDict['pool1'] = self.pool1


        self.conv21 = nn.Conv2d(
            in_channels = 64,
            out_channels = 128,
            kernel_size = 3,
            padding = 1
        )
        self.features.append(self.conv21)
        self.layerDict['conv21'] = self.conv21

        self.ReLU21 = nn.ReLU()
        self.features.append(self.ReLU21)
        self.layerDict['ReLU21'] = self.ReLU21

        self.conv22 = nn.Conv2d(
            in_channels = 128,
            out_channels = 128,
            kernel_size = 3,
            padding = 1
        )
        self.features.append(self.conv22)
        self.layerDict['conv22'] = self.conv22

        self.ReLU22 = nn.ReLU()
        self.features.append(self.ReLU22)
        self.layerDict['ReLU22'] = self.ReLU22

        self.pool2 = nn.MaxPool2d(2,2)
        self.features.append(self.pool2)
        self.layerDict['pool2'] = self.pool2

        self.conv31 = nn.Conv2d(
            in_channels = 128,
            out_channels = 128,
            kernel_size = 3,
            padding = 1
        )
        self.features.append(self.conv31)
        self.layerDict['conv31'] = self.conv31

        self.ReLU31 = nn.ReLU()
        self.features.append(self.ReLU31)
        self.layerDict['ReLU31'] = self.ReLU31

        self.conv32 = nn.Conv2d(
```

```
            in_channels = 128,
            out_channels = 128,
            kernel_size = 3,
            padding = 1
        )
        self.features.append(self.conv32)
        self.layerDict['conv32'] = self.conv32


        self.ReLU32 = nn.ReLU()
        self.features.append(self.ReLU32)
        self.layerDict['ReLU32'] = self.ReLU32

        self.pool3 = nn.MaxPool2d(2,2)
        self.features.append(self.pool3)
        self.layerDict['pool3'] = self.pool3

        self.classifier = []

        self.feature_dims = 4 * 4 * 128
        self.fc1 = nn.Linear(self.feature_dims, 512)
        self.classifier.append(self.fc1)
        self.layerDict['fc1'] = self.fc1

        self.fc1act = nn.Sigmoid()
        self.classifier.append(self.fc1act)
        self.layerDict['fc1act'] = self.fc1act

        self.fc2 = nn.Linear(512, 10)
        self.classifier.append(self.fc2)
        self.layerDict['fc2'] = self.fc2

    def forward(self, x):
        for layer in self.features:
            x = layer(x)

        x = x.view(-1, self.feature_dims)

        for layer in self.classifier:
            x = layer(x)
        return x
```

代码的结构比较简单。在定义网络(尤其是针对图像类任务)时一般为了使用清晰和方便都会将结构分为features和classifier两部分。features一般包括卷积、池化、正则化等非全连接层，classifier一般包括两到三个全连接层。这样网络结构比较清晰，也准确的描述了这些网络的功能。

main.py

```
# import *** omitted for simplicity
def setLearningRate(optimizer, lr):
    """Sets the learning rate to the initial LR decayed by 10 every 30 epochs"""
    for param_group in optimizer.param_groups:
        param_group['lr'] = lr
def train(DATASET = 'CIFAR10', NEpochs = 200,
        BatchSize = 32, learningRate = 1e-3, NDecreaseLR = 20, eps = 1e-3,
        AMSGrad = True, model_dir = "checkpoints/CIFAR10/", model_name =
"ckpt.pth", gpu = True):
```

```
    print("DATASET: ", DATASET)


    mu = torch.tensor([0.485, 0.456, 0.406], dtype=torch.float32)
    sigma = torch.tensor([0.229, 0.224, 0.225], dtype=torch.float32)
    Normalize = transforms.Normalize(mu.tolist(), sigma.tolist())
    Unnormalize = transforms.Normalize((-mu / sigma).tolist(), (1.0 /
sigma).tolist())
    tsf = {
        'train': transforms.Compose(
        [
        transforms.RandomHorizontalFlip(),
        transforms.RandomAffine(degrees = 10, translate = [0.1, 0.1], scale =
[0.9, 1.1]),
        transforms.ToTensor(),
        Normalize
        ]),
       'test': transforms.Compose(
        [
        transforms.ToTensor(),
        Normalize
        ])
        }
    trainset = torchvision.datasets.CIFAR10(root='./data/CIFAR10', train = True,
                                    download=True, transform = tsf['train'])
    testset = torchvision.datasets.CIFAR10(root='./data/CIFAR10', train = False,
                                    download=True, transform = tsf['test'])

    net = smallVGG(3)

    x_train, y_train = trainset.data, trainset.targets,
    x_test, y_test = testset.data, testset.targets,

    trainloader = torch.utils.data.DataLoader(trainset, batch_size = BatchSize,
                                    shuffle = True, num_workers = 1)
    testloader = torch.utils.data.DataLoader(testset, batch_size = 1000,
                                    shuffle = False, num_workers = 1)

    trainIter = iter(trainloader)
    testIter = iter(testloader)
    criterion = nn.CrossEntropyLoss()
    softmax = nn.Softmax(dim=1)

    if gpu:
        net.cuda()
        criterion.cuda()
        softmax.cuda()

    optimizer = optim.Adam(params = net.parameters(), lr = learningRate, eps =
eps, amsgrad = AMSGrad)

    NBatch = int(len(trainset) / BatchSize)
    cudnn.benchmark = True
    for epoch in range(NEpochs):
        lossTrain = 0.0
        accTrain = 0.0
        for i in range(NBatch):
```

```
            try:
                batchX, batchY = trainIter.next()
            except StopIteration:
                trainIter = iter(trainloader)
                batchX, batchY = trainIter.next()

            if gpu:
                batchX = batchX.cuda()
                batchY = batchY.cuda()

            optimizer.zero_grad()
            logits = net.forward(batchX)
            prob = softmax(logits)

            loss = criterion(logits, batchY)
            loss.backward()
            optimizer.step()

            lossTrain += loss.cpu().detach().numpy() / NBatch
            if gpu:
                pred = np.argmax(prob.cpu().detach().numpy(), axis = 1)
                groundTruth = batchY.cpu().detach().numpy()
            else:
                pred = np.argmax(prob.detach().numpy(), axis = 1)
                groundTruth = batchY.detach().numpy()

            acc = np.mean(pred == groundTruth)
            accTrain += acc / NBatch

        if (epoch + 1) % NDecreaseLR == 0:
            learningRate = learningRate / 2.0
            setLearningRate(optimizer, learningRate)
        accTest,lossTest = test(testloader, net, gpu=gpu)
        print("Epoch: ", epoch, "Train Loss: ", lossTrain, "Train accuracy: ",
accTrain*100, "Test Loss: ",\
              lossTest,"Test accuracy: ",accTest*100)

    if not os.path.exists(model_dir):
        os.makedirs(model_dir)
    torch.save(net, model_dir + model_name)
    print("Model saved")
def test(testloader, net,gpu = True):
    testIter = iter(testloader)
    acc = 0.0
    NBatch = 10
    cri = nn.CrossEntropyLoss()
    lossTest = 0.0
    for i, data in enumerate(testIter, 0):
        batchX, batchY = data
        if gpu:
            batchX = batchX.cuda()
            batchY = batchY.cuda()
        logits = net.forward(batchX)
        loss = cri(logits, batchY)
        lossTest += loss.cpu().detach().numpy() / NBatch
        if gpu:
            pred = np.argmax(logits.cpu().detach().numpy(), axis = 1)
            groundTruth = batchY.cpu().detach().numpy()
```

```
        else:
            pred = np.argmax(logits.detach().numpy(), axis = 1)
            groundTruth = batchY.detach().numpy()
        acc += np.mean(pred == groundTruth)
    accTest = acc / NBatch
    return accTest,lossTest
if __name__ == '__main__':
    import argparse
    import sys
    import traceback

    try:
        parser = argparse.ArgumentParser()
        parser.add_argument('--dataset', type = str, default = 'CIFAR10')
        parser.add_argument('--epochs', type = int, default = 100)
        parser.add_argument('--eps', type = float, default = 1e-3)
        parser.add_argument('--AMSGrad', type = bool, default = True)
        parser.add_argument('--batch_size', type = int, default = 32)
        parser.add_argument('--learning_rate', type = float, default = 1e-3)
        parser.add_argument('--decrease_LR', type = int, default = 20)

        parser.add_argument('--nogpu', dest='gpu', action='store_false')
        parser.set_defaults(gpu=True)
        args = parser.parse_args()

        model_dir = "checkpoints/" + args.dataset + '/'
        model_name = "ckpt.pth"
        train(DATASET = args.dataset, NEpochs = args.epochs,
        BatchSize = args.batch_size, learningRate = args.learning_rate,
NDecreaseLR = args.decrease_LR, eps = args.eps,
        AMSGrad = args.AMSGrad, model_dir = model_dir, model_name = model_name,
gpu = args.gpu)

    except:
        traceback.print_exc(file=sys.stdout)
        sys.exit(1)
```

这里也只是一个简单的模型的pipeline,没有涉及到复杂的操作。类似的简单模型都可以用这样的pipeline进行训练，只要定义好网络的接口即可。

结果展示

经过100epoches的训练，可以得到如下的结果

```
Epoch:  0 Train Loss:  1.7037 Train accuracy:  36.69 Test Loss:  1.2625 Test
accuracy:  54.31
Epoch:  1 Train Loss:  1.2118 Train accuracy:  55.91 Test Loss:  1.0374 Test
accuracy:  63.25
Epoch:  2 Train Loss:  0.9448 Train accuracy:  66.4 Test Loss:  0.7656 Test
accuracy:  73.14
Epoch:  3 Train Loss:  0.8016 Train accuracy:  71.78 Test Loss:  0.6717 Test
accuracy:  76.36
Epoch:  4 Train Loss:  0.7119 Train accuracy:  74.97 Test Loss:  0.6023 Test
accuracy:  79.53
Epoch:  5 Train Loss:  0.6424 Train accuracy:  77.54 Test Loss:  0.6355 Test
accuracy:  78.17
```

```
Epoch:  6 Train Loss:  0.5985 Train accuracy:  79.16 Test Loss:  0.5348 Test
accuracy:  81.79
Epoch:  7 Train Loss:  0.5644 Train accuracy:  80.49 Test Loss:  0.5145 Test
accuracy:  82.15
Epoch:  8 Train Loss:  0.5323 Train accuracy:  81.5 Test Loss:  0.516 Test
accuracy:  82.45
Epoch:  9 Train Loss:  0.5065 Train accuracy:  82.26 Test Loss:  0.4728 Test
accuracy:  83.59
Epoch:  10 Train Loss:  0.4923 Train accuracy:  82.89 Test Loss:  0.4893 Test
accuracy:  83.16
Epoch:  11 Train Loss:  0.479 Train accuracy:  83.33 Test Loss:  0.5008 Test
accuracy:  82.72
Epoch:  12 Train Loss:  0.4766 Train accuracy:  83.57 Test Loss:  0.4649 Test
accuracy:  84.02
Epoch:  13 Train Loss:  0.4672 Train accuracy:  84.0 Test Loss:  0.5326 Test
accuracy:  81.76
Epoch:  14 Train Loss:  0.4723 Train accuracy:  83.68 Test Loss:  0.4914 Test
accuracy:  83.52
Epoch:  15 Train Loss:  0.4803 Train accuracy:  83.37 Test Loss:  0.4824 Test
accuracy:  83.49
Epoch:  16 Train Loss:  0.4894 Train accuracy:  83.17 Test Loss:  0.4773 Test
accuracy:  83.65
Epoch:  17 Train Loss:  0.4991 Train accuracy:  82.66 Test Loss:  0.5199 Test
accuracy:  82.01
Epoch:  18 Train Loss:  0.5318 Train accuracy:  81.84 Test Loss:  0.5481 Test
accuracy:  81.46
Epoch:  19 Train Loss:  0.5613 Train accuracy:  80.59 Test Loss:  0.5441 Test
accuracy:  81.12
Epoch:  20 Train Loss:  0.4639 Train accuracy:  84.05 Test Loss:  0.4429 Test
accuracy:  84.99
Epoch:  21 Train Loss:  0.4343 Train accuracy:  85.02 Test Loss:  0.4456 Test
accuracy:  84.96
Epoch:  22 Train Loss:  0.4257 Train accuracy:  85.4 Test Loss:  0.4558 Test
accuracy:  84.4
Epoch:  23 Train Loss:  0.4136 Train accuracy:  85.66 Test Loss:  0.447 Test
accuracy:  85.19
Epoch:  24 Train Loss:  0.4029 Train accuracy:  86.24 Test Loss:  0.4494 Test
accuracy:  85.16
Epoch:  25 Train Loss:  0.4091 Train accuracy:  85.92 Test Loss:  0.4318 Test
accuracy:  85.16
Epoch:  26 Train Loss:  0.3998 Train accuracy:  86.02 Test Loss:  0.4106 Test
accuracy:  86.35
Epoch:  27 Train Loss:  0.3897 Train accuracy:  86.51 Test Loss:  0.4222 Test
accuracy:  85.76
Epoch:  28 Train Loss:  0.3928 Train accuracy:  86.29 Test Loss:  0.4308 Test
accuracy:  85.47
Epoch:  29 Train Loss:  0.3863 Train accuracy:  86.6 Test Loss:  0.4446 Test
accuracy:  84.9
Epoch:  30 Train Loss:  0.3822 Train accuracy:  86.87 Test Loss:  0.4364 Test
accuracy:  85.53
Epoch:  31 Train Loss:  0.3809 Train accuracy:  86.81 Test Loss:  0.4173 Test
accuracy:  85.95
Epoch:  32 Train Loss:  0.3899 Train accuracy:  86.5 Test Loss:  0.4429 Test
accuracy:  85.24
Epoch:  33 Train Loss:  0.3811 Train accuracy:  86.84 Test Loss:  0.4209 Test
accuracy:  85.9
Epoch:  34 Train Loss:  0.3792 Train accuracy:  86.88 Test Loss:  0.4112 Test
accuracy:  86.29
```

```
Epoch:  35 Train Loss:  0.3732 Train accuracy:  86.97 Test Loss:  0.4239 Test
accuracy:  85.81
Epoch:  36 Train Loss:  0.3745 Train accuracy:  87.06 Test Loss:  0.4159 Test
accuracy:  85.59
Epoch:  37 Train Loss:  0.3732 Train accuracy:  87.16 Test Loss:  0.446 Test
accuracy:  85.38
Epoch:  38 Train Loss:  0.3691 Train accuracy:  87.52 Test Loss:  0.4075 Test
accuracy:  86.34
Epoch:  39 Train Loss:  0.3676 Train accuracy:  87.17 Test Loss:  0.4343 Test
accuracy:  85.61
Epoch:  40 Train Loss:  0.3368 Train accuracy:  88.17 Test Loss:  0.3882 Test
accuracy:  87.01
Epoch:  41 Train Loss:  0.3208 Train accuracy:  88.99 Test Loss:  0.3807 Test
accuracy:  87.28
Epoch:  42 Train Loss:  0.3117 Train accuracy:  89.25 Test Loss:  0.38 Test
accuracy:  87.52
Epoch:  43 Train Loss:  0.3097 Train accuracy:  89.28 Test Loss:  0.3864 Test
accuracy:  87.08
Epoch:  44 Train Loss:  0.3113 Train accuracy:  89.13 Test Loss:  0.3729 Test
accuracy:  87.79
Epoch:  45 Train Loss:  0.3031 Train accuracy:  89.43 Test Loss:  0.3773 Test
accuracy:  87.49
Epoch:  46 Train Loss:  0.2984 Train accuracy:  89.67 Test Loss:  0.3685 Test
accuracy:  87.78
Epoch:  47 Train Loss:  0.2944 Train accuracy:  89.75 Test Loss:  0.3767 Test
accuracy:  87.74
Epoch:  48 Train Loss:  0.2945 Train accuracy:  89.78 Test Loss:  0.3803 Test
accuracy:  87.43
Epoch:  49 Train Loss:  0.2893 Train accuracy:  89.86 Test Loss:  0.3732 Test
accuracy:  87.29
Epoch:  50 Train Loss:  0.2882 Train accuracy:  89.98 Test Loss:  0.3679 Test
accuracy:  87.6
Epoch:  51 Train Loss:  0.2894 Train accuracy:  89.9 Test Loss:  0.3749 Test
accuracy:  87.52
Epoch:  52 Train Loss:  0.2903 Train accuracy:  89.87 Test Loss:  0.3691 Test
accuracy:  87.58
Epoch:  53 Train Loss:  0.2814 Train accuracy:  90.28 Test Loss:  0.3712 Test
accuracy:  87.93
Epoch:  54 Train Loss:  0.2797 Train accuracy:  90.25 Test Loss:  0.3594 Test
accuracy:  88.29
Epoch:  55 Train Loss:  0.286 Train accuracy:  90.08 Test Loss:  0.3728 Test
accuracy:  87.62
Epoch:  56 Train Loss:  0.2822 Train accuracy:  90.18 Test Loss:  0.3802 Test
accuracy:  87.53
Epoch:  57 Train Loss:  0.2804 Train accuracy:  90.21 Test Loss:  0.3805 Test
accuracy:  87.51
Epoch:  58 Train Loss:  0.2845 Train accuracy:  90.12 Test Loss:  0.3648 Test
accuracy:  87.86
Epoch:  59 Train Loss:  0.2829 Train accuracy:  90.14 Test Loss:  0.3912 Test
accuracy:  87.04
Epoch:  60 Train Loss:  0.2579 Train accuracy:  91.12 Test Loss:  0.3535 Test
accuracy:  88.33
Epoch:  61 Train Loss:  0.2546 Train accuracy:  91.16 Test Loss:  0.3488 Test
accuracy:  88.65
Epoch:  62 Train Loss:  0.247 Train accuracy:  91.42 Test Loss:  0.3557 Test
accuracy:  88.45
Epoch:  63 Train Loss:  0.2487 Train accuracy:  91.21 Test Loss:  0.352 Test
accuracy:  88.42
```

```
Epoch:  64 Train Loss:  0.2469 Train accuracy:  91.42 Test Loss:  0.3521 Test
accuracy:  88.62
Epoch:  65 Train Loss:  0.2453 Train accuracy:  91.49 Test Loss:  0.3528 Test
accuracy:  88.59
Epoch:  66 Train Loss:  0.2448 Train accuracy:  91.34 Test Loss:  0.3611 Test
accuracy:  88.27
Epoch:  67 Train Loss:  0.247 Train accuracy:  91.32 Test Loss:  0.3561 Test
accuracy:  88.38
Epoch:  68 Train Loss:  0.2457 Train accuracy:  91.43 Test Loss:  0.3546 Test
accuracy:  88.39
Epoch:  69 Train Loss:  0.2385 Train accuracy:  91.58 Test Loss:  0.3573 Test
accuracy:  88.26
Epoch:  70 Train Loss:  0.2389 Train accuracy:  91.75 Test Loss:  0.3538 Test
accuracy:  88.41
Epoch:  71 Train Loss:  0.2389 Train accuracy:  91.66 Test Loss:  0.3575 Test
accuracy:  88.33
Epoch:  72 Train Loss:  0.2335 Train accuracy:  91.89 Test Loss:  0.3627 Test
accuracy:  88.31
Epoch:  73 Train Loss:  0.2353 Train accuracy:  91.75 Test Loss:  0.3606 Test
accuracy:  88.36
Epoch:  74 Train Loss:  0.2304 Train accuracy:  91.98 Test Loss:  0.3528 Test
accuracy:  88.67
Epoch:  75 Train Loss:  0.234 Train accuracy:  91.86 Test Loss:  0.3533 Test
accuracy:  88.42
Epoch:  76 Train Loss:  0.2317 Train accuracy:  91.82 Test Loss:  0.3599 Test
accuracy:  88.24
Epoch:  77 Train Loss:  0.2314 Train accuracy:  91.93 Test Loss:  0.3532 Test
accuracy:  88.66
Epoch:  78 Train Loss:  0.2315 Train accuracy:  91.86 Test Loss:  0.3571 Test
accuracy:  88.65
Epoch:  79 Train Loss:  0.2268 Train accuracy:  92.17 Test Loss:  0.3482 Test
accuracy:  88.89
Epoch:  80 Train Loss:  0.2204 Train accuracy:  92.32 Test Loss:  0.349 Test
accuracy:  88.78
Epoch:  81 Train Loss:  0.219 Train accuracy:  92.4 Test Loss:  0.3467 Test
accuracy:  88.97
Epoch:  82 Train Loss:  0.2207 Train accuracy:  92.31 Test Loss:  0.3464 Test
accuracy:  89.02
Epoch:  83 Train Loss:  0.2178 Train accuracy:  92.48 Test Loss:  0.3458 Test
accuracy:  88.94
Epoch:  84 Train Loss:  0.2162 Train accuracy:  92.38 Test Loss:  0.3443 Test
accuracy:  88.88
Epoch:  85 Train Loss:  0.216 Train accuracy:  92.42 Test Loss:  0.3442 Test
accuracy:  88.97
Epoch:  86 Train Loss:  0.2137 Train accuracy:  92.56 Test Loss:  0.3485 Test
accuracy:  88.85
Epoch:  87 Train Loss:  0.2156 Train accuracy:  92.46 Test Loss:  0.3473 Test
accuracy:  88.9
Epoch:  88 Train Loss:  0.2152 Train accuracy:  92.52 Test Loss:  0.3468 Test
accuracy:  88.92
Epoch:  89 Train Loss:  0.2132 Train accuracy:  92.62 Test Loss:  0.3479 Test
accuracy:  88.93
Epoch:  90 Train Loss:  0.2097 Train accuracy:  92.64 Test Loss:  0.3512 Test
accuracy:  88.66
Epoch:  91 Train Loss:  0.2138 Train accuracy:  92.46 Test Loss:  0.3498 Test
accuracy:  88.71
Epoch:  92 Train Loss:  0.2111 Train accuracy:  92.69 Test Loss:  0.3437 Test
accuracy:  88.79
```

```
Epoch:  93 Train Loss:  0.212 Train accuracy:  92.5 Test Loss:  0.3494 Test
accuracy:  88.54
Epoch:  94 Train Loss:  0.2153 Train accuracy:  92.45 Test Loss:  0.3438 Test
accuracy:  89.06
Epoch:  95 Train Loss:  0.2077 Train accuracy:  92.76 Test Loss:  0.349 Test
accuracy:  88.88
Epoch:  96 Train Loss:  0.2065 Train accuracy:  92.82 Test Loss:  0.3456 Test
accuracy:  88.82
Epoch:  97 Train Loss:  0.2068 Train accuracy:  92.64 Test Loss:  0.3443 Test
accuracy:  88.96
Epoch:  98 Train Loss:  0.2063 Train accuracy:  92.73 Test Loss:  0.3466 Test
accuracy:  88.83
Epoch:  99 Train Loss:  0.2035 Train accuracy:  92.9 Test Loss:  0.3447 Test
accuracy:  89.03
```

注意到我们的训练并没有引起严重的过拟合。整体来generalization-gap处于可接受的范围。结果使用pycharm的自动保存输出在output.pdf中。

结果分析与结论

- 模型比较小，结果也比较合理，没有很多复杂的改进和设计
- 后面希望用NNI调优可以取得更好的效果
- 优化器的选取要进行慎重考虑，尤其注意使用weight decay, weight decay可以有效避免训练结果的"抖动"
- 注意要避免过拟合，这里使用了数据增强的方法和weight decay，之前做别的项目的时候，试过对CIFAR10不使用数据增强，结果造成严重的过拟合。对于CIFAR10这种较小的数据集尤其要注意。