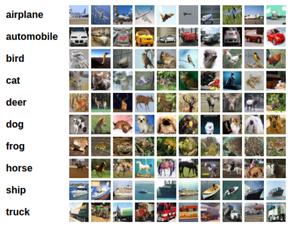
数据

torchvision

对于视觉任务，pytorch定义了package包，以进行部分数据的加载（Imagenet, CIFAR10, MNIST, etc. ）
CIFAR10数据集所包含的类别: 'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'.

图像大小都为"3*32*32"大小



数据加载

torchvision加载数据后返回是 PILImage images of range [0, 1]，我们需要将其转换成Tensor，并归一化（normalize）到[−1,1]

```
import torch
import torchvision
import torchvision.transforms as transforms
#每个通道都要归一化
trainform=transforms.Compose([transforms.ToTensor,transforms.Normalize((0.5,0.5,0.5),
(0.5,0.5,0.5))])#transform的定义

trainset=torchvision.datasets.CIFAR10(root='./',train=True,download=True,transform=trainsform)
trainloader=torch.utils.data.DataLoader(trainset,batch_size=4,shuffle=True,num_workers=2)
testset=torchvision.datasets.CIFAR10(root='./',train=False,download=True,transform=trainsform)
testloader=torch.utils.data.DataLoader(testset,batch_size=4,shuffle=False,num_workers=2)

classes=('plane', 'car', 'bird', 'cat','deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

模型定义

和minist的模型一样，只是输入数据的通道数变了

```
class CifarModel(nn.Module):
    def __init__(self):
        super(MinistModel,self).__init__()
        self.conv1=nn.Conv2d(3,6,5)
        self.conv2=nn.Conv2d(6,16,5)
        self.fc1=nn.Linear(16*5*5,120)
        self.fc2=nn.Linear(120,84)
        self.fc3=nn.Linear(84,10)
    def num_flat_features(self,x):#在这里就是16*5*5
        size=x.shape[1:]#except batch size
        num_features=1
        for s in size:
            num_features*=s
        return num_features
    def forward(self, x):
        x=F.max_pool2d(F.relu(self.conv1(x)),(2,2))
        x=F.max_pool2d(F.relu(self.conv2(x)),(2,2))
        x=x.view(-1,self.num_flat_features(x))
        x=F.relu(self.fc1(x))
        x=F.relu(self.fc2(x))
        return self.fc3(x)
```

网络训练

```


net=CifarModel()
net=net.to(device)

optimizer=optim.SGD(net.parameters(),lr=0.0005,momentum=0.9)
criterion=nn.CrossEntropyLoss()

def test():
    results=[]
    labels=[]

    # for evry category
    class_correct = np.array(list(0. for i in range(10)))
    class_total = np.array(list(0. for i in range(10)))

    for i,data in enumerate(testloader):
        img,label=data
        labels.extend(label)
        output=net(img.to(device))

        _, predicted = torch.max(output, 1)
        # print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
        #                               for j in range(4)))
        results.extend(predicted.cpu())

    # 计算准确率、精度、召回
    results=np.array(results)
    labels=np.array(labels)
    acc=np.equal(results,labels).sum()/results.shape[0]
    TP=((labels+results)==2).sum()
    FP=((labels-results)==-1).sum()
    FN=((labels-results)==1).sum()
    TN=((labels+results)==0).sum()
    rec=TP/(FN+TP)
    precision=TP/(FP+TP)
    print('epoch %d: accuracy: %3f---precision: %.3f---recall: %.3f' % (epoch + 1, acc, precision, rec))

    #计算每个类的准确率
    c = (results == labels).squeeze()
    for i in range(labels.shape[0]):
        class_correct[labels[i]] += c[i].item()
        class_total[labels[i]] += 1
    for i in range(10):
        print('Accuracy of %5s : %2d %%' % (
            classes[i], 100 * class_correct[i] / class_total[i]))



#train the network
EPOCH=100
for epoch in range(EPOCH):
    running_loss=0
    for i,data in enumerate(trainloader):
        img,label=data
        output=net(img.to(device))
        loss=criterion(output,label.to(device))
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        running_loss+=loss.cpu()
        if(i%2000==1999):
            print('[%d,%5d] loss: %.3f' % (epoch+1,i+1,running_loss/2000))
            running_loss=0
        pass
    if((epoch+1)%10==0):
        print("testing.........")
        test()
        torch.save(net.state_dict(),os.path.join("./pretrained","epoch-"+str(epoch+1)+".pkl"))

print('Finished Training')
```

实验效果

```
epoch 60: accuracy: 0.603900---precision: 0.658---recall: 0.705
Accuracy of plane : 66 %
Accuracy of   car : 71 %
Accuracy of  bird : 43 %
Accuracy of   cat : 42 %
Accuracy of  deer : 50 %
Accuracy of   dog : 46 %
Accuracy of  frog : 65 %
Accuracy of horse : 68 %
Accuracy of  ship : 71 %
Accuracy of truck : 78 %
```

感觉这个模型不是很适合用在这个数据集上