浙江大学实验报告

姓名：林炬乙　　学号：3180103721

课程名称：数字图像处理　　任课老师：项志宇

实验名称：cifar-10 物品分类识别　　实验日期：2021/6/23

1 实验目的和要求

（分点简要说明本次实验需要进行的工作和最终的目的）
实现一个 CNN，进行 CIFAR-10 物品分类识别。

2 实验原理

CIFAR-10 是一个接近普适物体的彩色图像数据集。CIFAR-10 是由 Hinton 的学生 Alex Krizhevsky 和 Ilya Sutskever 整理的一个用于识别普适物体的小型数据集。一共包含 10 个类别的 RGB 彩色图片：飞机（ airplane ）、汽车（ automobile ）、鸟类（ bird ）、猫（ cat ）、鹿（ deer ）、狗（ dog ）、蛙类（ frog ）、马（ horse ）、船（ ship ）和卡车（ truck ）.

每个图片的尺寸为 32 × 32 ，每个类别有 6000 个图像，数据集中一共有 50000 张训练图片和 10000 张测试图片。CIFAR-10 是 32*32 的彩色图， 直接读入官网的数据就是处理好的 dict 格式.如果没有, 可以利用 numpy 的 frombuffer 读入,可能还需要 reshape 处理. 相比于手写字符， CIFAR-10 含有的是现实世界中真实的物体，不仅噪声很大，而且物体的比例、特征都不尽相同，这为识别带来很大困难。直接的线性模型如 Softmax 在 CIFAR-10 上表现得很差。

3 实验内容

3.1 加载和初始化训练和测试的数据集

自己定义数据集, 转换、归一化并载入，我们将它们转换为归一化范围[-1，1]的张量, 展示一些训练图像，展示的同时会显示物品类别.

```
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')  # dict 有四个 key，其中 data 是 10000*6072 的图片数据，labels
    return dict  # 是标成 0-9 分类，还有其他 key 有字符串格式的分类说明
```

```
def load_data(folder, data_name):
    data_batch1 = unpickle(folder + data_name)
    train_set_x = data_batch1[b'data']
    train_set_x = train_set_x.reshape(10000, 3, 32, 32)  # 转成矩阵格式
    train_set_x = np.transpose(train_set_x, (0, 2, 3, 1))  # 改成 10000
* 32 * 32* 3
    binary_set_x = train_set_x / 255  #归一化
    train_set_y = data_batch1[b'labels']
    train_set_y = np.array(train_set_y) #载入 labels
    return binary_set_x, train_set_y
```

自己定义数据集

```
class DealDataset(Dataset):

    def __init__(self, folder, data_name, transform=None):
        (train_set, train_labels) = load_data(folder, data_name)  # 其
实也可以直接使用 torch.load(),读取之后的结果为 torch.Tensor 形式
        self.train_set = train_set
        self.train_labels = train_labels
        self.transform = transform

    def __getitem__(self, index):
        img, target = self.train_set[index],
int(self.train_labels[index])
        if self.transform is not None:
            img = self.transform(img)
        return img, target

    def __len__(self):
        return len(self.train_set)
```

转换、归一化并载入，我们将它们转换为归一化范围[-1，1]的张量：

```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

train_dataset = DealDataset(r'./cifar/', "data_batch_1",
transform=transform)
test_dataset = DealDataset(r'./cifar/', "test_batch",
transform=transform)

train_loader = torch.utils.data.DataLoader(train_dataset,
batch_size=BATCH_SIZE, shuffle=True)
test_loader = torch.utils.data.DataLoader(train_dataset,
batch_size=BATCH_SIZE, shuffle=False)
```

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

展示一些训练图像，展示的同时会显示物品类别：

```
# 图片可视化
images, labels = next(iter(train_loader))
img = torchvision.utils.make_grid(images)
img = img / 2 + 0.5
img = img.numpy().transpose(1, 2, 0)
print(' '.join('%5s' % classes[labels[j]] for j in range(8)))
plt.imshow(img)
plt.show()
```

2. 定义卷积神经网络和损失函数

对网络进行一定的修改，以获取三通道图像：

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        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 forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
```

```
        x = self.fc3(x)
        return x

net = Net()
net = net.double()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.002, momentum=0.9) #优化
器
```

3. 根据训练数据训练网络

在实际运行中，我发现物品分类需要比较多的训练次数才有比较好的效果，这里取了 epoch=50。同样会 在训练结束后绘制 loss 曲线。

4. 在测试数据上测试网络

我们先试试测试一个 batch（这里 batch=8）的数据，并且把它显示出来看看效果。选一个 batch 的测试 数据，输出它的对应类别并显示出来，然后用我们的网络测试它，输出分类结果并对比。

```
# 测试
images, labels = next(iter(test_loader))
img = torchvision.utils.make_grid(images)
img = img / 2 + 0.5
img = img.numpy().transpose(1, 2, 0)
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in
range(8)))
plt.imshow(img)
plt.show()

net = Net().double()
net.load_state_dict(torch.load(PATH))

outputs = net(images)

_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(8)))
```

然后测试整个测试数据集的结果并输出，这里我输出了整个数据集的测试准确率以及每个类别的准确 率：

```
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))

class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in test_loader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))
```

4 实验结果和分析

（使用图片和文字叙述实验结果，并对这些结果进行适当分析）
载入数据时，可视化一部分图片：
输出类别：

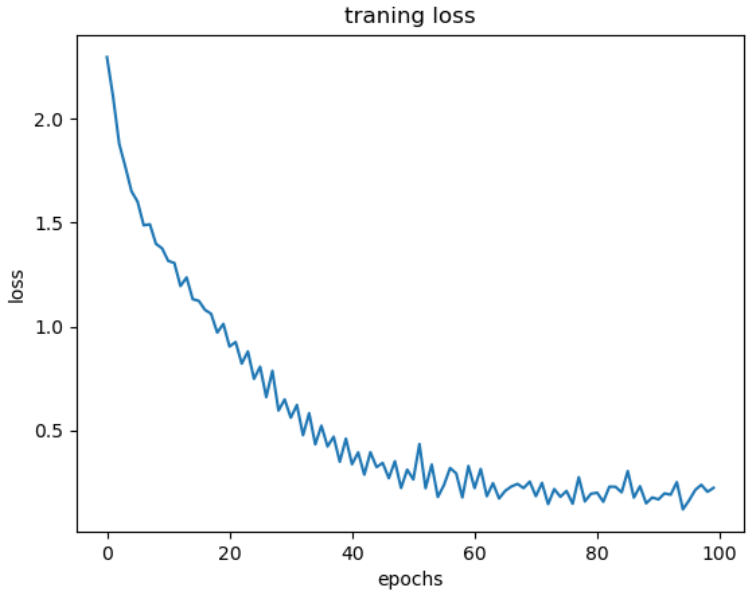
```
C:\ProgramData\Miniconda3\python.exe "C:/Users/12638/Deskto
  dog plane horse   car   cat plane   dog  deer
```

显示图片，仔细观察可以和类别对上：

训练过程截图：

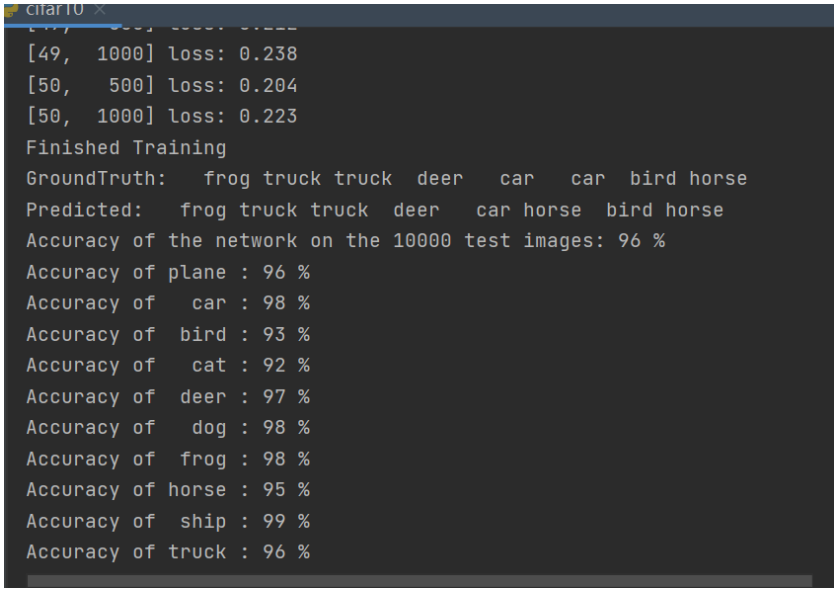
```
[4,  1000] loss: 1.492
[5,   500] loss: 1.397
[5,  1000] loss: 1.376
[6,   500] loss: 1.315
[6,  1000] loss: 1.306
[7,   500] loss: 1.194
[7,  1000] loss: 1.236
[8,   500] loss: 1.131
[8,  1000] loss: 1.123
[9,   500] loss: 1.080
[9,  1000] loss: 1.061
```

loss 的变化曲线：



测试结果输出如下，其中第一行是测试样本对应分类，第二行是我们的模型对测试样本的分类。第三行 是我们的模型对 10000 个测试图片的测试准确率，并且给出了所有类别的测试

准确率。

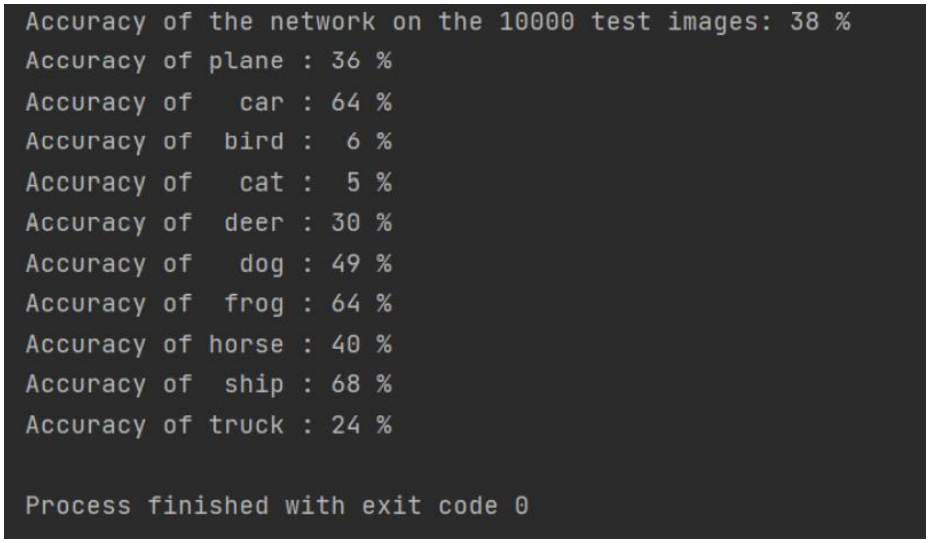


可以看到我们达到了 96%的准确率，其中轮船识别率最高，猫的识别率最低。

编程体会:

搭建网络、训练和测试的部分，网上能找到很多有关资料，学起来很快，但我觉得最有意思的是我自己 实现了数据的加载，实现了自己定义的数据集。这部分网上能查到的资料比较少，大部分都是直接用给 定的接口做了。但比起直接用 Pytorch 已经弄好的数据接口读入，自己尝试着去读更有意思，也加深了 我对它的理解。特别是，一开始我以为 CIFAR 是需要转成灰度图的，查了很多资料应该怎么读入数据转 成灰度，做到后面发现 CIFAR 就应该用彩色图做，当时顿时觉得有点无奈，但回过头来看，我在查怎么转成灰度的时候，也加深了我对这个数据结构的了解。

另外有一点很有意思的是，CIFAR 我最后是跑了 50 个 epoch 达到 94%的准确率，其中汽车的识别率最 低，猫的识别率最高。但是一开始，我只跑了 2 个 epoch 试试看的时候，它的结果其实是这样的:



这个时候猫的正确率其实是很低的，而汽车却有比较好的识别效果。观察了一部分测试数据和结果之 后，我认为，除了训练随机性的差别以外，汽车的整体特征其实只需要比较少的

训练次数就能大概得 到，所以在 epoch=2 的时候就能有 64%；而猫的图片很多都是一个模糊的比较小的形状，刚开始训练的时候是很难提取到特征的。但在 epoch 足够大的时候，比如我们最后用 epoch=50，猫的特征能够捕捉 完全，正确率有很大提高，最后达到 92%；汽车的一部分图片却很容易和卡车混起来，它们的某些特征是相似的，具体到一些特殊的图片更是如此.