LeNet5 实现对 CIFAR10 的分类

18052223 饶晓龙

目录

1.数据读取

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（ arplane ）、汽车（ automobile ）、鸟类（ bird ）、猫（ cat ）、鹿（ deer ）、狗（ dog ）、蛙类（ frog ）、马（ horse ）、船（ ship ）和卡车（ truck ）。图片的尺寸为 32×32 ，数据集中一共有 50000 张训练圆片和 10000 张测试图片。

与 MNIST 数据集中目比， CIFAR-10 具有以下不同点：

• CIFAR-10 是 3 通道的彩色 RGB 图像，而 MNIST 是灰度图像。

• CIFAR-10 的图片尺寸为 32×32， 而 MNIST 的图片尺寸为 28×28，比 MNIST 稍大。

• 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。

首先使用 torchvision 加载和归一化我们的训练数据和测试数据。

torchvision 这个东西，实现了常用的一些深度学习的相关的图像数据的加载功能，比如 cifar10、Imagenet、Mnist 等等的，保存在 torchvision.datasets 模块中。

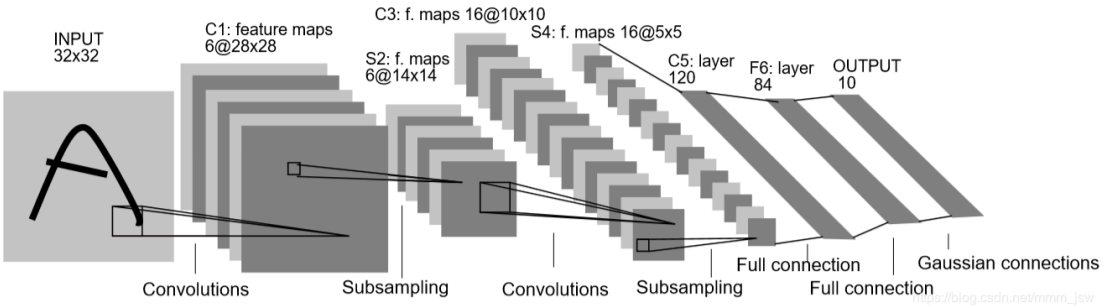
同时，也封装了一些处理数据的方法。保存在 torchvision.transforms 模块中

还封装了一些模型和工具封装在相应模型中,比如 torchvision.models 当中就包含了 AlexNet，VGG，ResNet，SqueezeNet 等模型。

由于 torchvision 的 datasets 的输出是[0,1]的 PILImage,所以我们先先归一化为[-1,1]的 Tensor 首先定义了一个变换 transform，利用的是上面提到的 transforms 模块中的 Compose()把多个变换组合在一起,可以看到这里面组合了 ToTensor 和 Normalize 这两个变换

transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))前面的(0.5,0.5,0.5）是 RGB 三个通道上的均值，后面(0.5,0.5,0.5)是三个通道的标准差，注意通道顺序是 RGB，因为 openCV 读出来的图像是 BRG 顺序。这两个 tuple 数据是用来对 RGB 图像做归一化的，如其名称 Normalize 所示这里都取 0.5 只是一个近似的操作，实际上其均值和方差并不是这么多，但是就这个示例而言影响可不计。精确值是通过分别计算 R,G,B 三个通道的数据算出来的。

2.LeNet5 网络结构



LeNet5 网络虽然很小，但是包含了深度学习的基本模块：卷积层、池化层、全连接层。LeNet5 共有七层，不包含输入，每层都包含可训练参数，每个层有多个 Feature Map，每个 Feature Map 通过一种卷积滤波器提取输入的一种特征，然后每 Feature Map 有多个神经元。

2.1 各层结构及参数

1. INPUT（输入层）

32*32 的图片，共有 1024 个神经元。

2.C1(卷积层)

选取 6 个 5*5 卷积核(不包含偏置),得到 6 个特征图，每个特征图的大小为 32-5+1=28，也就是神经元的个数由 1024 减小到了 28*28=784。输入层与 C1 层之间的参数：6*(5*5+1)，对于卷积层 C1，每个像素都与前一层的 5*5 个像素和

1 一个 bias 有连接，有 6*(5*5+1)*(28*28)个连接。

3.S2（池化层）

池化层是一个下采样层，有 6 个 14*14 的特征图，特征图中的每个单元与 C1 中相对应特征图的 2*2 邻域连接。S2 层每个单元对应 C1 中 4 个求和，乘以一个可训练参数，再加上一个可训练偏置。

C1 与 S2 之间的参数：每一个 2*2 求和，然后乘以一个参数，加上一个偏置，共计 2*6=12 个参数。S2 中的每个像素都与 C1 中的 2*2 个像素和 1 个偏置相连接，所以有 6*5*14*14=5880 个链接。

4.C3（卷积层）

选取卷积核大小为 5*5，得到新的图片大小为 10*10。

S2 与 C3 之间的组合：如图所示前 6 个 feature map 与 S2 层相连的 3 个 feature map 相连接，后面 6 个 feature map 与 S2 层相连的 4 个 feature map 相连接，后面 3 个 feature map 与 S2 层部分不相连的 4 个 feature map 相连接，最后一个与 S2 层的所有 feature map 相连。卷积核的大小依然为 5*5，总共有 6*(3*5*5+1)+6*(4*5*5+1)+3*(4*5*5+1)+1*(6*5*5+1)=1516 个参数。而图像大小为 10*10，所以共有 151600 个连接。

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

5.S4（池化层）

窗口大小为 2*2，有 16 个特征图，共有 32 个参数。

C3 与 S4 之间的参数： 16*(25*4+25)=200016*(25*4+25)=2000 个连接。

6.C5（卷积层）

总共 120 个 feature map，每个 feature map 与 S4 层所有的 feature map 相连接，卷积核大小为 5*5，而 S4 层的 feature map 的大小也是 5*5，所以 C5 的 feature map 就变成了一个点，共计有 120*(25*16+1)=48120 个参数。

7.F6（全连接层）

F6 相当于 MLP（Multi-Layer Perceptron，多层感知机）中的隐含层，有 84 个节点，所以有 84*(120+1)=10164 个参数，F6 采用了 sigmoid 函数。

8.Output（输出层）

全连接层，共有 10 个节点，采用的是径向基函数（RBF）的网络连接方式。

总结：

1. 卷积神经网络能够很好的利用图像的结构信息。

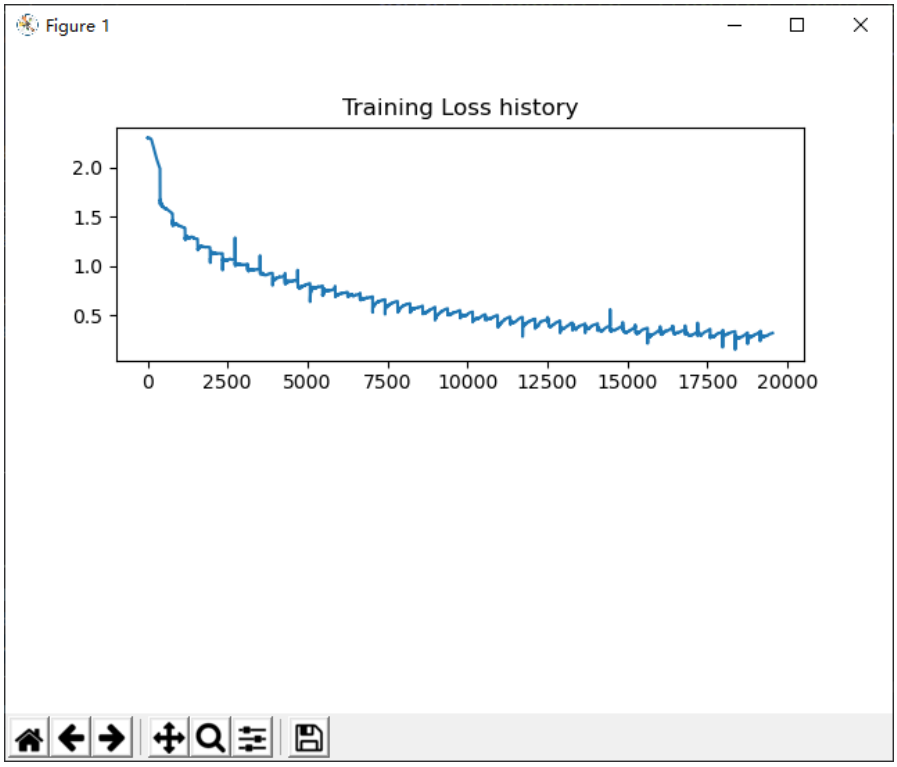
2. 卷积层的参数较少，这也是由卷积层的主要特性即局部连接和共享权重所决定。

3.训练过程及预测结果展示

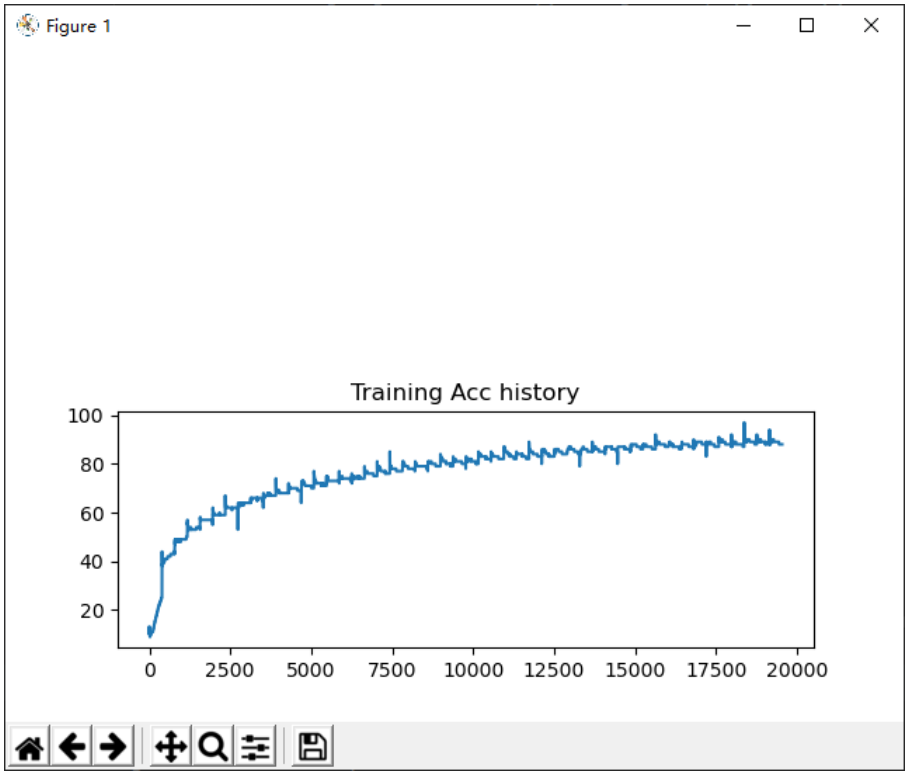
训练完成结果:

```
LeNet_CA10
[epoch:50, iter:19536] Loss: 0.305 | Acc: 88.000%
[epoch:50, iter:19537] Loss: 0.306 | Acc: 88.000%
[epoch:50, iter:19538] Loss: 0.306 | Acc: 88.000%
[epoch:50, iter:19539] Loss: 0.306 | Acc: 88.000%
[epoch:50, iter:19540] Loss: 0.306 | Acc: 88.000%
[epoch:50, iter:19541] Loss: 0.307 | Acc: 88.000%
[epoch:50, iter:19542] Loss: 0.307 | Acc: 88.000%
[epoch:50, iter:19543] Loss: 0.307 | Acc: 88.000%
[epoch:50, iter:19544] Loss: 0.307 | Acc: 88.000%
[epoch:50, iter:19545] Loss: 0.307 | Acc: 88.000%
[epoch:50, iter:19546] Loss: 0.307 | Acc: 88.000%
[epoch:50, iter:19547] Loss: 0.307 | Acc: 88.000%
[epoch:50, iter:19548] Loss: 0.307 | Acc: 88.000%
[epoch:50, iter:19549] Loss: 0.308 | Acc: 88.000%
[epoch:50, iter:19550] Loss: 0.308 | Acc: 88.000%
Finished Training
```

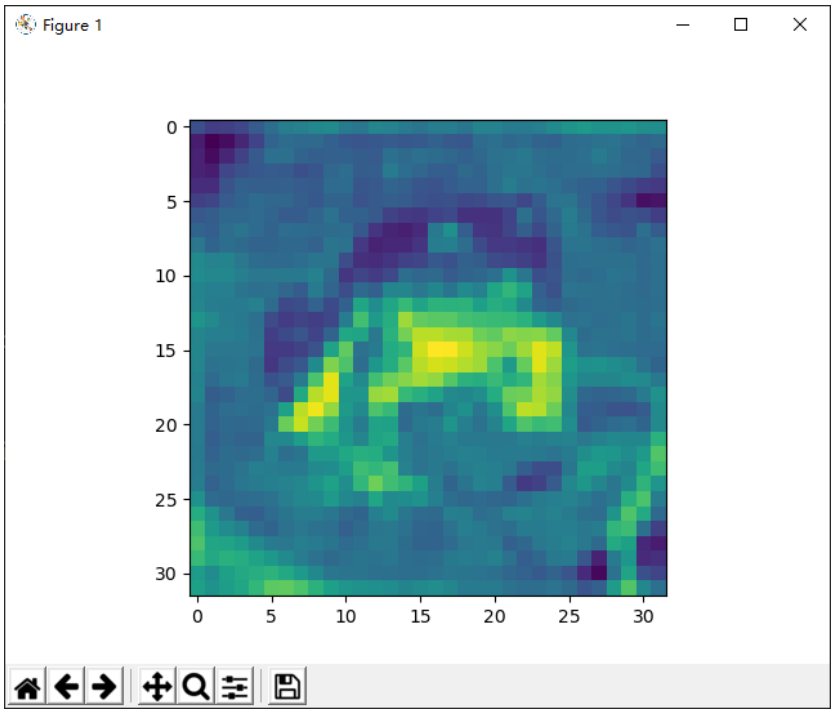
Loss 曲线:



Acc 曲线:



预测单张图片结果




```
概率 tensor([[0.0960, 0.1003, 0.0903, 0.0929, 0.0960, 0.1021, 0.1051, 0.1049, 0.1069,
         0.1054]], device='cuda:0', grad_fn=<SoftmaxBackward>)
类别 8
tensor([0.0778], device='cuda:0')
分类 ship

Process finished with exit code 0
```

4.源代码

LeNet_CA10.py　　//使用 LeNet5 训练 CIFAR10

```
import torch
import torchvision
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
from torch import nn
from LeNet5 import net
from torch.autograd import Variable
import torch.optim as optim

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


# datasets.CIFAR10( )也是封装好了的，就在 torchvision.datasets 块中

trainset = datasets.CIFAR10(root='D:\DeepLearning\cs231n-cnn-master\cs231n-cnn-master', train=True,download=True, transform=transform)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=128,shuffle=True, num_workers=2)


# 对于测试集的操作和训练集一样

testset = torchvision.datasets.CIFAR10(root='D:\DeepLearning\cs231n-cnn-master\cs231n-cnn-master', train=False,download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=128,shuffle=False, num_workers=2)

    # 类别信息也是需要我们给定的

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


#用到了神经网络工具箱 nn 中的交叉熵损失函数

criterion = nn.CrossEntropyLoss()

# 使用 SGD（随机梯度下降）优化，学习率为 0.001，动量为 0.9

optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
```

```
def train():
    plotloss = []
    plotauc = []
    for epoch in range(50):  # 指定训练一共要循环几个 epoch
        net.train()
        sum_loss = 0.0
        correct = 0.0
        total = 0.0

    # 这里我们遇到了第一步中出现的 trailoader，代码传入数据，enumerate 是 python 的内置函数，既获得索引也获得数据
        for i, (images, labels) in enumerate(trainloader):
            # data 是从 enumerate 返回的 data，包含数据和标签信息，分别赋值给 inputs 和 labels
            # data 的结构是：[4x3x32x32 的张量,长度 4 的张量],4 是 batch_size 的数值

            # 把 input 数据从 tensor 转为 variable，variable 才拥有梯度 grad,输入模型训练都要转成 Variable
            if torch.cuda.is_available():
                images = Variable(images).cuda()
                labels = Variable(labels).cuda()
            else:
                images = Variable(images)
                labels = Variable(labels)

        # 将参数的 grad 值初始化为
            optimizer.zero_grad()

        # forward + backward + optimize
            outputs = net(images)
        # 将 output 和 labels 使用叉熵计算损失
```

```
            loss = criterion(outputs, labels)
        # 反向传播
            loss.backward()
        # 用 SGD 更新参数
            optimizer.step()

        # loss.item()转换为 numpy
        # loss 本身为 Variable 类型，所以要使用 loss.data[0]获取其 Tensor，因为其为标量，所以取 0
            sum_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)  # 更新测试图片的数量
            correct += (predicted == labels).sum()  # 更新正确分类的图片的数量
        #             if i % 200 == 199:
            print('[epoch:%d, iter:%d] Loss: %.03f | Acc: %.3f%% '
                  % (epoch + 1, (i + 1 + epoch * len(trainloader)), sum_loss / (i + 1), 100. * correct / total))
            plotloss.append(sum_loss / (i + 1))
            plotauc.append(100. * correct / total)

    print('Finished Training')
    plt.subplot(2, 1, 1)
    plt.plot(plotloss)
    plt.title('Training Loss history')
    plt.show()
    plt.subplot(2, 1, 2)
    plt.plot(plotauc)
    plt.title('Training Acc history')
    plt.show()

'''
def test():
    class_correct = list(0. for i in range(10))
    class_total = list(0. for i in range(10))
    for data in testloader:
```

```
            images, labels = data
            images = Variable(images).cuda()
            labels = Variable(labels).cuda()
            outputs = net(images)

            _, predicted = torch.max(outputs.data, 1)
            # 4 组(batch_size)数据中，输出于 label 相同的，标记为 1，否则为 0
            c = (predicted == labels).squeeze()
            for i in range(16):
                label = labels[i]   # 对各个类的进行各自累加
                class_correct[label] += c[i]
                class_total[label] += 1

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

if __name__ == '__main__':
    train()
    #test()
    #torch.save(the_model.state_dict(), 'D:\DeepLearning\cs231n-cnn-
master\cs231n-cnn-master/CIFAR-10/model/LeNet5-128.pth')
```

LeNet5.py　　//LeNet5 网络框架

```
import torch
import torch.nn as nn

# 若能使用 cuda，则使用 cuda
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')


# 定义网络
class LeNet5(nn.Module):  # nn.Module 是所有神经网络的基类，我们自己定义任
何神经网络，都要继承 nn.Module
    def __init__(self):
        super(LeNet5, self).__init__()
```

```
        self.conv1 = nn.Sequential(
            # 卷积层 1，3 通道输入，6 个卷积核，核大小 5*5
            # 经过该层图像大小变为 32-5+1，28*28
            nn.Conv2d(in_channels=3，out_channels=6，kernel_size=5，stride=1,
padding=0),
            # 激活函数
            nn.ReLU(),
            # 经 2*2 最大池化，图像变为 14*14
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0),
        )
        self.conv2 = nn.Sequential(
            # 卷积层 2，6 输入通道，16 个卷积核，核大小 5*5
            # 经过该层图像变为 14-5+1，10*10
            nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5, stride=1,
padding=0),
            nn.ReLU(),
            # 经 2*2 最大池化，图像变为 5*5
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0),
        )
        self.fc = nn.Sequential(
            # 接着三个全连接层
            nn.Linear(16 * 5 * 5, 120),
            nn.ReLU(),
            nn.Linear(120, 84),
            nn.ReLU(),
            nn.Linear(84, 10),
        )

        # 定义前向传播过程，输入为

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        # nn.Linear()的输入输出都是维度为一的值，所以要把多维度的 tensor 展
```

平成一维

```
        x = x.view(x.size()[0], -1)
        x = self.fc(x)
        return x


net = LeNet5().cuda()
print("LeNet5 out: ", net)
torch.save(net,        'D:\DeepLearning\cs231n-cnn-master\cs231n-cnn-master/CIFAR-10/model/LeNet5-128.pth')
```

image.py　　//CIFAR10 可视化

```
# 用于将 cifar10 的数据可视化

import pickle as p
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as plimg
from PIL import Image


def load_CIFAR_batch(filename):
    with open(filename, 'rb')as f:
        #            datadict = p.load(f)
        datadict = p.load(f, encoding='latin1')
        X = datadict['data']
        Y = datadict['labels']
        X = X.reshape(10000, 3, 32, 32)
        Y = np.array(Y)
        return X, Y


def load_CIFAR_Labels(filename):
    with open(filename, 'rb') as f:
        lines = [x for x in f.readlines()]
        print(lines)


if __name__ == "__main__":
    load_CIFAR_Labels("D:\DeepLearning\cs231n-cnn-master\cs231n-cnn-master\cifar-10-batches-py/batches.meta")
```

```
    imgX, imgY = load_CIFAR_batch("D:\DeepLearning\cs231n-cnn-master\cs231n-cnn-master\cifar-10-batches-py/data_batch_1")
    print(imgX.shape)
    print("正在保存图片:")
    #     for i in range(imgX.shape[0]):
    for i in range(10):   # 值输出 10 张图片，用来做演示
        #             imgs = imgX[i - 1]#?
        imgs = imgX[i]
        img0 = imgs[0]
        img1 = imgs[1]
        img2 = imgs[2]
        i0 = Image.fromarray(img0)   # 从数据，生成 image 对象
        i1 = Image.fromarray(img1)
        i2 = Image.fromarray(img2)
        img = Image.merge("RGB", (i0, i1, i2))
        name = "img" + str(i) + ".png"
        img.save("D:\DeepLearning\cs231n-cnn-master\cs231n-cnn-master\cifar10_images/" + name, "png")   # 文件夹下是 RGB 融合后的图像

        for j in range(imgs.shape[0]):
            #                         img = imgs[j - 1]
            img = imgs[j]
            name = "img" + str(i) + str(j) + ".png"
            print("正在保存图片" + name)
            plimg.imsave("D:\DeepLearning\cs231n-cnn-master\cs231n-cnn-master\cifar10_images/" + name, img)   # 文件夹下是 RGB 分离的图像
    print("保存完毕.")
```

predication.py //使用训练好的模型来进行单张图片的预测

```
import torch
from PIL import Image
from torch.autograd import Variable
import torch.nn.functional as F
from torchvision import datasets, transforms
import numpy as np
import matplotlib.pyplot as plt
```

```
import matplotlib.image as mpimg
import torch
import torch.nn as nn

# 若能使用 cuda，则使用 cuda
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')


# 定义网络
class LeNet5(nn.Module):  # nn.Module 是所有神经网络的基类，我们自己定义任何神经网络，都要继承 nn.Module
    def __init__(self):
        super(LeNet5, self).__init__()
        self.conv1 = nn.Sequential(
            # 卷积层 1，3 通道输入，6 个卷积核，核大小 5*5
            # 经过该层图像大小变为 32-5+1，28*28
            nn.Conv2d(in_channels=3, out_channels=6, kernel_size=5, stride=1, padding=0),
            # 激活函数
            nn.ReLU(),
            # 经 2*2 最大池化，图像变为 14*14
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0),
        )
        self.conv2 = nn.Sequential(
            # 卷积层 2，6 输入通道，16 个卷积核，核大小 5*5
            # 经过该层图像变为 14-5+1，10*10
            nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5, stride=1, padding=0),
            nn.ReLU(),
            # 经 2*2 最大池化，图像变为 5*5
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0),
        )
        self.fc = nn.Sequential(
```

```
            # 接着三个全连接层
            nn.Linear(16 * 5 * 5, 120),
            nn.ReLU(),
            nn.Linear(120, 84),
            nn.ReLU(),
            nn.Linear(84, 10),
        )

        # 定义前向传播过程，输入为

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        # nn.Linear()的输入输出都是维度为一的值，所以要把多维度的 tensor 展平成一维
        x = x.view(x.size()[0], -1)
        x = self.fc(x)
        return x


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

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = torch.load('D:\DeepLearning\cs231n-cnn-master\cs231n-cnn-master/CIFAR-10/model/LeNet5-128.pth')  # 加载模型
model = model.to(device)
model.eval()  # 把模型转为 test 模式

# 读取要预测的图片
img = Image.open("D:\DeepLearning\cs231n-cnn-master\cs231n-cnn-master\cifar10_images/img01.png").convert('RGB')  # 读取图像
trans = transforms.Compose([transforms.Resize((32, 32)),
                            transforms.ToTensor(),
                            transforms.Normalize(mean=(0.5, 0.5, 0.5),
```

```
                                                    std=(0.5, 0.5, 0.5)),
                                ])
lena = mpimg.imread('D:\DeepLearning\cs231n-cnn-master\cs231n-cnn-master\cifar10_images/img01.png')

lena.shape  #(512, 512, 3)
plt.imshow(lena)  # 显示图片

#plt.axis('off')  # 不显示坐标轴
plt.show()
img = trans(img)
img = img.to(device)
# 图片扩展多一维,因为输入到保存的模型中是 4 维的[batch_size,通道,长，宽],
而普通图片只有三维，[通道,长，宽]
img = img.unsqueeze(0)
# 扩展后，为[1，1，28，28]
output = model(img)
prob = F.softmax(output, dim=1)  # prob 是 10 个分类的概率
print("概率", prob)
value, predicted = torch.max(output.data, 1)
print("类别", predicted.item())
print(value)
pred_class = classes[predicted.item()]
print("分类", pred_class)
```