作业三：风格迁移 —— 自由发挥

## 小组情况：

小组成员：穆新宇，付光明，蒋任驰，董若扬

小组成员分工：

## 使用的方法：

导入了d2l包中的绘图类，方便展示运行过程中损失的变换。

定义两个距离，一个用于内容Dc和一个用于样式 Ds。Dc测量两个图像之间的内容差异，同时Ds测量两个图像之间的样式差异。然后，获取第三个图像，即输入，并对其进行转换，以最小化其与内容图像的内容距离和与样式图像的样式距离。

内容损失函数表示加权后单层内容与原输入间的距离，该函数采用特征图*FXL*​的图层*L*在网络处理输入中*X*并返回加权内容*X*和*C*之间距离*wCL*​\**DCL*​(*X*，*C*)。内容图像的特征图（*FCL*​）。将此函数实现为torch模块，其构造函数为*FCL*​作为输入。距||*FXL*​−*FCL*||2是两组特征映射之间的均方误差，可以使用nn.MSELoss 计算。

样式丢失模块的实现方式与内容丢失模块类似。它将充当网络中的透明层，用于计算该层的样式损失。为了计算样式损失，需要计算gram矩阵*GXL*​.格拉姆矩阵是将给定矩阵乘以其转置矩阵的结果。

必须通过将每个元素除以矩阵中的元素总数来规范化格拉姆矩阵。这种规范化是为了抵消以下影响： *F*^*XL*​具有很大的矩阵维度N，在 Gram 矩阵中产生较大的值。这些较大的值将导致第一层（在池化层之前）在梯度下降期间产生更大的影响，而样式特征往往位于网络的更深层。

PyTorch的VGG实现是一个模块，分为两个Sequential 子模块：features （卷积层和池化层）和classifier （完全连接层）。这里使用features 模块，因为需要各个卷积层的输出来测量内容和样式损失。某些层在训练期间的行为与评估时的行为不同，因此必须使用.eval()将网络设置为评估模式。VGG网络在图像上进行训练，每个通道归一化为meanst=[0.485， 0.456， 0.406]和std=[0.229， 0.224， 0.225]。在将图像输入到网络之前，将使用它们对其进行规范化。

使用L-BFGS算法来运行梯度下降。与训练网络不同，这里希望训练输入图像，以尽量减少内容/样式损失。将创建一个PyTorch L-BFGS优化器optim.LBFGS，并将图像作为要优化的张量传递给它。

最后，定义一个执行神经传递的函数。对于网络的每次迭代，它都会获得更新的输入并计算新的损失。我们将运行每个损失模块的方法来动态计算它们的梯度。优化器需要一个“闭包”函数，该函数重新评估模块并返回损失。网络可能会尝试使用超过图像的 0 到 1 张量范围的值来优化输入，可以通过在每次运行网络时将输入值更正为0到1来解决此问题。

## 自己写的代码段用灰色标注

from \_\_future\_\_ import print\_function  
!pip install d2l == 0.17.5.  
% matplotlib inline  
from d2l import torch as d2l  
  
import torch  
import torch.nn as nn  
import torch.nn.functional as func  
import torch.optim as optim  
from PIL import Image *# 加载显示图片*import matplotlib.pyplot as plt  
import torchvision.transforms as transforms *# 将PIL图像转为张量*import torchvision.models as models *#加载预先训练的模型*import os  
from google.colab import drive *# 读取云端硬盘文件*

device = torch.device*(*"cuda" if torch.cuda.is\_available*()* else "cpu"*)*

image\_size = 1024 if torch.cuda.is\_available*()* else 128 *# 如果没有GPU，就使用较小的图像尺寸*load = transforms.Compose*([* transforms.Resize*(*image\_size*)*, *# 修改图片尺寸* transforms.ToTensor*()]) # 转成torch tensor格式*def image\_loader*(*image\_name*)*:  
 image = Image.open*(*image\_name*)  
 # 拟合网络输入尺寸所需的额外批维度* image = load*(*image*)*.unsqueeze*(*0*)* return image.to*(*device, torch.float*)*

*# 从云端硬盘读取输入*drive.mount*(*'/content/drive'*)*path = "/content/drive/My Drive"  
  
os.chdir*(*path*)*os.listdir*(*path*)*

*# 内容图像和风格图像*content\_img = image\_loader*(*"dancing.jpg"*)*style\_img = image\_loader*(*"picasso.jpg"*)*

plt.ion*()*def imshow*(*tensor, title=None*)*:  
 img = tensor.cpu*()*.clone*() # 在副本上操作* img = img.squeeze*(*0*) # 移除此前设置的额外维度* img = transforms.ToPILImage*()(*img*)* plt.imshow*(*img*)* if title is not None:  
 plt.title*(*title*)* plt.pause*(*0.001*) # 暂停一会等待绘图完成*plt.figure*()*imshow*(*content\_img, title='Content Image'*)*plt.figure*()*imshow*(*style\_img, title='Style Image'*)*

class ContentLoss*(*nn.Module*)*: *# 内容损失* def \_\_init\_\_*(*self, target*)*:  
 super*(*ContentLoss, self*)*.\_\_init\_\_*()* self.loss = None  
 self.target = target.detach*()* def forward*(*self, input*)*:  
 loss = func.mse\_loss*(*input, self.target*)* self.loss = loss  
 return input

def gram\_matrix*(*input*)*: *# 格拉姆矩阵* channels, n = input.shape*[*1*]*, input.numel*()* // input.shape*[*1*]* input = input.reshape*((*channels, n*))* return torch.matmul*(*input, input.T*)* / *(*channels \* n*)*

class StyleLoss*(*nn.Module*)*: *#风格损失* def \_\_init\_\_*(*self, target\_feature*)*:  
 super*(*StyleLoss, self*)*.\_\_init\_\_*()* self.loss = None  
 self.target = gram\_matrix*(*target\_feature*)*.detach*()* def forward*(*self, input*)*:  
 g = gram\_matrix*(*input*)* loss=func.mse\_loss*(*g, self.target*)* self.loss = loss  
 return input

*# 导入一个预先训练的神经网络（19层VGG网络）*cnn = models.vgg19*(*pretrained=True*)*.features.to*(*device*)*.eval*()*

norm\_mean = torch.tensor*([*0.485, 0.456, 0.406*])*.to*(*device*)*norm\_std = torch.tensor*([*0.229, 0.224, 0.225*])*.to*(*device*)*class Normalization*(*nn.Module*)*: *# 标准化以便在网络上操作* def \_\_init\_\_*(*self, mean, std*)*:  
 super*(*Normalization, self*)*.\_\_init\_\_*()  
 # .view()使其变成 [C x 1 x 1] 以便于直接和 [B x C x H x W] 的图像张量操作  
 # B(batch size)批处理大小. C(channels) 通道数. H(height) 高. W(width) 宽.* self.mean = torch.tensor*(*mean*)*.view*(*-1, 1, 1*)* self.std = torch.tensor*(*std*)*.view*(*-1, 1, 1*)* def forward*(*self, img*)*:  
 *# 标准化图像张量* return *(*img - self.mean*)* / self.std

*# 计算 内容/风格 损失 :*content\_layers\_default = *[*'conv\_4'*]*style\_layers\_default = *[*'conv\_1', 'conv\_2', 'conv\_3', 'conv\_4', 'conv\_5'*]*def get\_style\_model\_and\_losses*(*cnn, norm\_mean, norm\_std,  
 style\_img, content\_img,  
 content\_layers=content\_layers\_default,  
 style\_layers=style\_layers\_default*)*:  
 *# 损失* content\_losses = *[]* style\_losses = *[]  
  
 # 标准化模块* normalization = Normalization*(*norm\_mean, norm\_std*)*.to*(*device*)  
  
 # 假定cnn是nn.Sequential* model = nn.Sequential*(*normalization*)* i = 0 *# 每次遇到卷积层时+1* for layer in cnn.children*()*:  
 if isinstance*(*layer, nn.Conv2d*)*:  
 i += 1  
 name = 'conv\_{}'.format*(*i*)* elif isinstance*(*layer, nn.ReLU*)*:  
 name = 'relu\_{}'.format*(*i*)  
 # 原地操作不适用于内容损失函数和风格损失函数，所以这里采用异地操作* layer = nn.ReLU*(*inplace=False*)* elif isinstance*(*layer, nn.MaxPool2d*)*:  
 name = 'pool\_{}'.format*(*i*)* elif isinstance*(*layer, nn.BatchNorm2d*)*:  
 name = 'bn\_{}'.format*(*i*)* else:  
 raise RuntimeError*(*'Unrecognized layer: {}'.format*(*layer.\_\_class\_\_.\_\_name\_\_*))* model.add\_module*(*name, layer*)* if name in content\_layers:  
 *# add content loss:* target = model*(*content\_img*)*.detach*()* content\_loss = ContentLoss*(*target*)* model.add\_module*(*"content\_loss\_{}".format*(*i*)*, content\_loss*)* content\_losses.append*(*content\_loss*)* if name in style\_layers:  
 *# add style loss:* target\_feature = model*(*style\_img*)*.detach*()* style\_loss = StyleLoss*(*target\_feature*)* model.add\_module*(*"style\_loss\_{}".format*(*i*)*, style\_loss*)* style\_losses.append*(*style\_loss*)  
  
 # 最后一次计算完内容损失和风格损失后裁剪图层* for i in range*(*len*(*model*)* - 1, -1, -1*)*:  
 if isinstance*(*model*[*i*]*, ContentLoss*)* or isinstance*(*model*[*i*]*, StyleLoss*)*:  
 break  
  
 model = model*[*:*(*i + 1*)]* return model, style\_losses, content\_losses

input\_img = content\_img.clone*()*plt.figure*()*imshow*(*input\_img, title='Input Image'*)*

def run\_style\_transfer*(*cnn, normalization\_mean, normalization\_std,  
 content\_img, style\_img, input\_img, num\_steps=1000,  
 style\_weight=1000000, content\_weight=1*)*:  
*# 使用d2l包里的作图工具*  
 animator = d2l.Animator*(*xlabel='epoch', ylabel='loss',  
 xlim=*[*50, num\_steps*]*,  
 legend=*[*'content', 'style', 'TV'*]*,  
 ncols=1, figsize=*(*7, 2.5*))* """开始运行风格迁移."""  
 model, style\_losses, content\_losses = get\_style\_model\_and\_losses*(*cnn,  
 normalization\_mean, normalization\_std, style\_img,  
 content\_img*)  
  
 # 需要优化输入的拷贝而不是模型的参数，所以根据需要更新所有的梯度* input\_img.requires\_grad\_*(*True*)* model.requires\_grad\_*(*False*)* optimizer = optim.LBFGS*([*input\_img*])* run = *[*0*]* while run*[*0*]* <= num\_steps:  
  
 def closure*()*:  
 *# 更正更新后的输入* with torch.no\_grad*()*:  
 input\_img.clamp\_*(*0, 1*)* optimizer.zero\_grad*()* model*(*input\_img*)* style\_score = 0  
 content\_score = 0  
  
 for sl in style\_losses:  
 style\_score += sl.loss  
 for cl in content\_losses:  
 content\_score += cl.loss  
  
 style\_score \*= style\_weight  
 content\_score \*= content\_weight  
  
 loss = style\_score + content\_score  
 loss.backward*()* run*[*0*]* += 1  
*# 作图*  
 if *(*run*[*0*]* + 1*)* % 10 == 0 and run*[*0*]* >= 49:  
 animator.add*(*run*[*0*]*, *[*float*(*content\_score.item*())*,  
 float*(*style\_score.item*())*, float*(*loss.item*())])* return style\_score + content\_score  
  
 optimizer.step*(*closure*)  
  
 # 最后还需修正* with torch.no\_grad*()*:  
 input\_img.clamp\_*(*0, 1*)* return input\_img

output = run\_style\_transfer*(*cnn, norm\_mean, norm\_std,  
 content\_img, style\_img, input\_img*)*plt.figure*()*imshow*(*output, title='Output Image'*)*plt.ioff*()*plt.show*()*