Style Transfer

一张是内容图像，另一张是风格图像。

将一个图像中的风格应用在另一图像之上，即风格迁移。

比如说一张相机拍的风景作为内容，梵高的油画作为风格

将风景照片转化成梵高油画风便叫做风格迁移。

Code

```
import torch
import torchvision
from torch import nn
from PIL import Image
from d2l import torch as d2l
import matplotlib.pyplot as plt
from torchvision.models import VGG19_Weights

d2l.set_figsize()
content_img = Image.open('Images/rainier.png').convert('RGB')
style_img = Image.open('Images/autumn.png').convert('RGB')
plt.imshow(content_img)
plt.show()
plt.imshow(style_img)
plt.show()

rgb_mean = torch.tensor([0.485, 0.456, 0.406])
rgb_std = torch.tensor([0.229,0.224,0.225])

def preprocess(img, image_shape):
    transforms = torchvision.transforms.Compose([
        torchvision.transforms.Resize(image_shape),
        torchvision.transforms.ToTensor(),
        torchvision.transforms.Normalize(mean=rgb_mean, std=rgb_std)
    ])
    return transforms(img).unsqueeze(0)

# 后处理函数，将tensor转化为图片
def postprocess(img):
    img = img[0].to(rgb_std.device) # img is a tensor
    # permute B C H W -> B H W C
    img = torch.clamp(img.permute(1,2,0) * rgb_std + rgb_mean, 0, 1)
    # B H W C -> B C H W , Tensor -> PIL
    return torchvision.transforms.ToPILImage()(img.permute(2,0,1))

# pretrained=True已经启用，所以使用weights
pretrained_net = torchvision.models.vgg19(weights=VGG19_Weights.IMAGENET1K_V1)

style_layers, content_layers = [0, 5, 10, 19, 28], [25]

net = nn.Sequential(*[pretrained_net.features[i]
                      for i in range(max(content_layers + style_layers) + 1)])

def extract_features(X, content_layers, style_layers):
```

```
    contents = []
    styles = []
    # 对于网络中的每一层
    for i in range(len(net)):
        # 在该层上运行输入X以提取特征
        X = net[i](X)  # 每一层抽特征
        if i in style_layers: # 如果该层为样式层，则返回样式
            styles.append(X)
        if i in content_layers: # 如果该层为内容层，则返回内容
            contents.append(X)
    return contents, styles

def get_contents(image_shape, device):
    content_X = preprocess(content_img, image_shape).to(device)
    # 从内容图像中提取内容特征
    content_Y, _ = extract_features(content_X, content_layers, style_layers)
    return content_X, content_Y

def get_styles(image_shape, device):
    style_X = preprocess(style_img, image_shape).to(device)
    # 从样式图像中提取样式特征
    _, styles_Y = extract_features(style_X, content_layers, style_layers)
    return style_X, styles_Y

def content_loss(Y_hat, Y):
    return torch.square(Y_hat - Y.detach()).mean()

def gram(X):
    # 计算通道数和特征数
    num_channels, n = X.shape[1], X.numel() // X.shape[1]
    # 将输入矩阵reshape为(通道数，特征数)的格式
    X = X.reshape((num_channels, n))
    return torch.matmul(X, X.T) / (num_channels * n)

def style_loss(Y_hat, gram_Y):
    return torch.square(gram(Y_hat) - gram_Y.detach()).mean()

def tv_loss(Y_hat):
    return 0.5 * (torch.abs(Y_hat[:,:,1:,:] - Y_hat[:,:,:-1,:]).mean()+
                  torch.abs(Y_hat[:,:,:,1:] - Y_hat[:,:,:,:-1]).mean())

# 风格转移的损失函数是内容损失、风格损失和总变化损失的加权和
# 定义内容损失、样式损失和总变差损失的权重
content_weight, style_weight, tv_weight = 1, 1e3, 10

# 总损失函数
def compute_loss(X, contents_Y_hat, styles_Y_hat, contents_Y, styles_Y_gram):
    contents_l = [
        content_loss(Y_hat, Y) * content_weight
        for Y_hat, Y in zip(contents_Y_hat, contents_Y) ]
    styles_l = [
        style_loss(Y_hat, Y) * style_weight
        for Y_hat, Y in zip(styles_Y_hat, styles_Y_gram) ]
    tv_l = tv_loss(X) * tv_weight
    l = sum(10 * styles_l + contents_l + [tv_l])
    return contents_l, styles_l, tv_l, l

# 生成合成图像
```

```
class SynthesizedImage(nn.Module):
    def __init__(self, img_shape, **kwargs):
        super(SynthesizedImage, self).__init__(**kwargs)
        self.weight = nn.Parameter(torch.rand(*img_shape))

    def forward(self):
        return self.weight

# 生成图像和优化器
def get_inits(X, device, lr, styles_Y):
    gen_img = SynthesizedImage(X.shape).to(device)
    gen_img.weight.data.copy_(X.data)
    trainer = torch.optim.Adam(gen_img.parameters(),lr=lr)
    styles_Y_gram = [gram(Y) for Y in styles_Y]
    return gen_img(), styles_Y_gram, trainer

def train(X, contents_Y, styles_Y, device, lr, num_epochs, lr_decay_epoch):
    X, styles_Y_gram, trainer = get_inits(X, device, lr, styles_Y)
    scheduler = torch.optim.lr_scheduler.StepLR(trainer, lr_decay_epoch, 0.8)
    animator = d2l.Animator(xlabel='epoch',ylabel='loss',xlim=[10,num_epochs],
                            legend=['content','style','TV'],ncols=2,figsize=(7, 2.5))
    for epoch in range(num_epochs):
        trainer.zero_grad()
        contents_Y_hat, styles_Y_hat = extract_features(X, content_layers,
style_layers)
        contents_l, styles_l, tv_l, l = compute_loss(X, contents_Y_hat, styles_Y_hat,
contents_Y, styles_Y_gram)
        l.backward()
        trainer.step()
        scheduler.step()
        if(epoch+1) % 10 == 0:
            animator.axes[1].imshow(postprocess(X))
            animator.add(epoch + 1,
                         [float(sum(contents_l)),
                          float(sum(styles_l)),
                          float(tv_l)])
    return X

device, image_shape = d2l.try_gpu(), (300,450)
net = net.to(device)
content_X, contents_Y = get_contents(image_shape, device)
_, styles_Y = get_styles(image_shape, device)
output = train(content_X, contents_Y, styles_Y, device, 0.3, 500, 50)
plt.show(output)
```