计算机科学与技术学院神经网络与深度学习课程实验报告

实验题目：风格迁移		学号：201600301304
日期：2019/4/10	班级： 人工智能 16	姓名： 贾乘兴
Email：1131225623@qq.com		
实验目的：实现 neural style transform 算法，并进行测试		
实验软件和硬件环境： 操作系统 mac os，内存 16GB，编译器 pycharm		

实验原理和方法：

1.neural style transform 算法

利用 vgg19，将一个内容图像和一个风格图像作为输入，返回一个按照所选择的风格图像加工的内容图像。

我们定义两个距离，一个用于内容（Dc），另一个用于（Ds）。Dc 测量两个图像的内容有多像，Ds 测量两个图像的风格有多像。然后我们采用一个新图像（例如一个噪声图像），对它进行变化，同时最小化与内容图像的距离和与风格图像的距离。

距离定义的公式如下

$$\mathcal{L}_{content}(\vec{p},\vec{x},l)=\frac{1}{2}\sum_{i,j}\left(F_{ij}^{l}-P_{ij}^{l}\right)^{2}$$

$$E_{l}=\frac{1}{4N_{l}^{2}M_{l}^{2}}\sum_{i,j}\left(G_{ij}^{l}-A_{ij}^{l}\right)^{2}$$

$$\mathcal{L}_{style}(\vec{a},\vec{x})=\sum_{l=0}^{L}w_{l}E_{l}$$

$$\mathcal{L}_{total}(\vec{p},\vec{a},\vec{x})=\alpha\mathcal{L}_{content}(\vec{p},\vec{x})+\beta\mathcal{L}_{style}(\vec{a},\vec{x})$$

对于内容图像，我们直接计算特征图的均方误差，对于风格图像，我们定义了 gram 矩阵，计算 gram 矩阵的均方误差，对于计算的层，内容我们计算的是第四层卷积层的结果，风格我们计算的是第一层第二层第三层第四层卷积层的结果

实验步骤：（不要求罗列完整源代码）
实验代码如下

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable
import torchvision
from torchvision import transforms, models
from PIL import Image
import argparse
import numpy as np
import os
import copy

#定义加载图像函数，并将PIL image转化为Tensor
use_gpu = torch.cuda.is_available()
dtype = torch.cuda.FloatTensor if use_gpu else torch.FloatTensor


def load_image(image_path, transforms=None, max_size=None, shape=None):
    image = Image.open(image_path)
    image_size = image.size

    if max_size is not None:
        #获取图像size，为sequence
        image_size = image.size
        #转化为float的array
        size = np.array(image_size).astype(float)
        size = max_size / size * size
        image = image.resize(size.astype(int), Image.ANTIALIAS)

    if shape is not None:
        image = image.resize(shape, Image.LANCZOS)

    #必须提供transform.ToTensor，转化为4D Tensor
    if transforms is not None:
        image = transforms(image).unsqueeze(0)

    #是否拷贝到GPU
    return image.type(dtype)

# 定义VGG模型，前向时抽取0,5,10,19,28层卷积特征
class VGGNet(nn.Module):
    ######################################
```

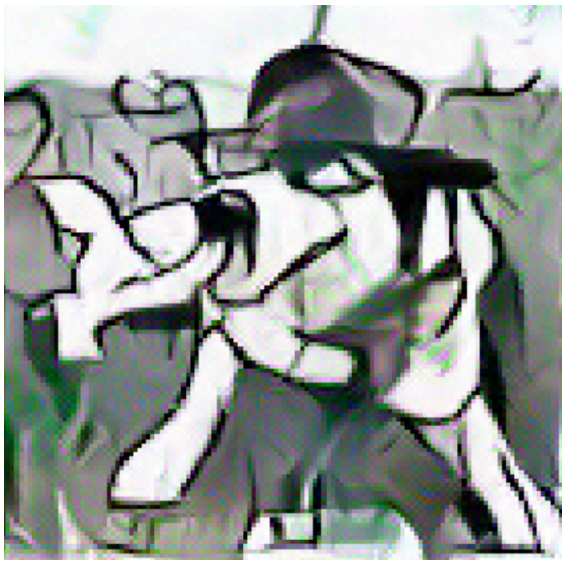
```
    #write your code
    def __init__(self):
        super(VGGNet, self).__init__()
        self.vgg = models.vgg19(pretrained=True).features
        self.content_layers_default = ['conv_4']
        self.style_layers_default = ['conv_1', 'conv_2', 'conv_3', 'conv_4',
'conv_5']
        self.model = nn.Sequential()

    def forward(self, img):
        cnn = copy.deepcopy(self.vgg)
        style_layers = self.style_layers_default

        model = nn.Sequential()  # the new Sequential module network
        features = []
        i = 1
        for layer in list(cnn):
            if(isinstance(layer, nn.Conv2d)):
                name = "conv_" + str(i)
                model.add_module(name, layer)
                if name in style_layers:
                    features.append(model(img).clone())
            if(isinstance(layer, nn.ReLU)):
                name = "relu_" + str(i)
                model.add_module(name, layer)
                i += 1
            if isinstance(layer, nn.MaxPool2d):
                name = "pool_" + str(i)
                model.add_module(name, layer)

        return features

    ######################################
#定义主函数
def main(config):
    #定义图像变换操作，必须定义.ToTensor()。（可做）
    transform = transforms.Compose(
        [transforms.ToTensor(),
        transforms.Normalize((0.485, 0.456, 0.406),
                             (0.229, 0.224, 0.225))
        ])

    #content 和 style 图像，style 图像 resize 成同样大小
    content = load_image(config.content, transform, max_size = config.max_size)
```

```
    style = load_image(config.style, transform, shape = [content.size(2),
content.size(3)])

    #将concent复制一份作为target，并需要计算梯度，作为最终的输出
    target = Variable(content.clone(), requires_grad = True)
    optimizer = torch.optim.Adam([target], lr = config.lr, betas=[0.5, 0.999])

    vgg = VGGNet()
    if use_gpu:
        vgg = vgg.cuda()

    for step in range(config.total_step):
        #分别计算target、content、style的特征图
        target_features = vgg(target)
        content_features = vgg(Variable(content))
        style_features = vgg(Variable(style))

        content_loss = 0.0
        style_loss = 0.0

        time = 0
        optimizer.zero_grad()

        for f1, f2, f3 in zip(target_features, content_features, style_features):
            pass
            #计算content_loss
            ########################################
            # write your code
            time += 1
            if time == 4:
                content_loss += F.mse_loss(f1, f2)
            ########################################

            #将特征reshape成二维矩阵相乘，求gram矩阵
            ########################################
            # write your code
            def GM(input):
                a, b, c, d = input.size()
                features = input.view(a * b, c * d)
                G = torch.mm(features, features.t())
                return G.div(a * b * c * d)

            gf1 = GM(f1)
            gf3 = GM(f3)
```

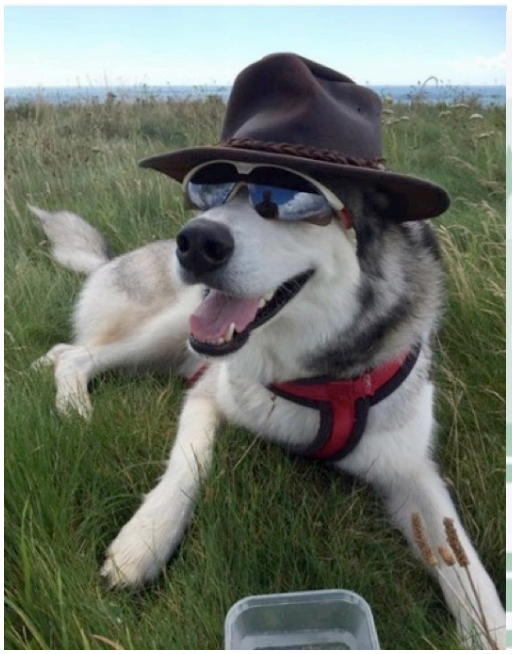
```
            ######################################


            #计算style_loss
            ######################################
            # write your code
            style_loss += F.mse_loss(gf1, gf3)
            ######################################


        #计算总的loss
        ######################################
        # write your code
        loss = config.style_weight * style_loss + content_loss
        print("iter %d" % step)
        print(style_loss.data)
        print(content_loss.data)
        print(loss.data)
        ######################################

        #反向求导与优化
        ######################################
        # write your code
        loss.backward()
        optimizer.step()
        ######################################

        if (step+1) % config.log_step == 0:
            print ('Step [%d/%d], Content Loss: %.4f, Style Loss: %.4f'
                  %(step+1, config.total_step, content_loss.data,
style_loss.data))

        if (step+1) % config.sample_step == 0:
            # Save the generated image
            denorm = transforms.Normalize((-2.12, -2.04, -1.80), (4.37, 4.46,
4.44))
            img = target.clone().cpu().squeeze()
            img = denorm(img.data).clamp_(0, 1)
            torchvision.utils.save_image(img, 'output-%d.png' %(step+1))

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('--content', type=str, default='./content.jpg')
    parser.add_argument('--style', type=str, default='./style.jpg')
```

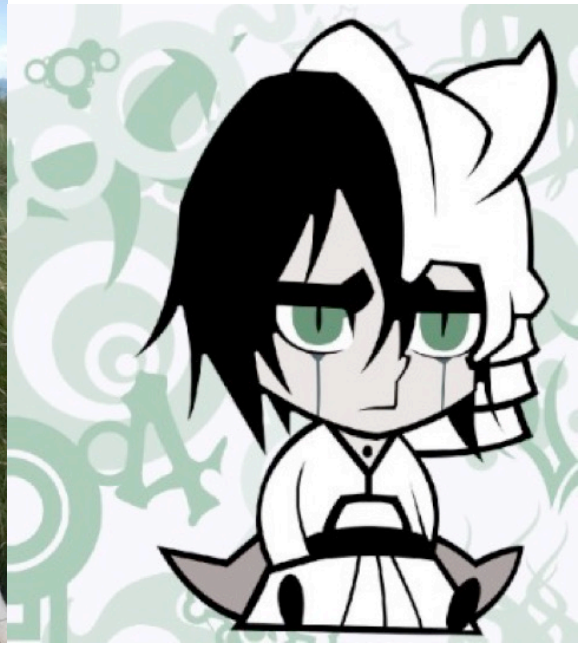
```
    parser.add_argument('--max_size', type=int, default=400)
    parser.add_argument('--total_step', type=int, default=1000)
    parser.add_argument('--log_step', type=int, default=10)
    parser.add_argument('--sample_step', type=int, default=50)
    parser.add_argument('--style_weight', type=float, default=1000)
    parser.add_argument('--lr', type=float, default=0.003)
    config = parser.parse_args()
    print(config)
    main(config)
```

结论分析与体会：
最终生成图像如下



内容图与风格图如下





最终的 style—loss 为 6.0533，content—loss 为 5.3921（sum）
就实验过程中遇到和出现的问题，你是如何解决和处理的，自拟 1－3 道问答题： 1. 最开始设置 weight 为 100，效果非常不好，跑出来的结果和原来基本一样，后来将 weight 改为 1000，效果较原来有了提升，而且 style—loss 下降快了很多 2. 尝试将 adam 的优化改为 LBFGS