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.select = ['0', '5', '10', '19', '28']

    def forward(self, img):
        features = []

        for name,layer in self.vgg._modules.items():
            img = layer(img)
            if name in self.select:
                features.append(img)

        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))
```

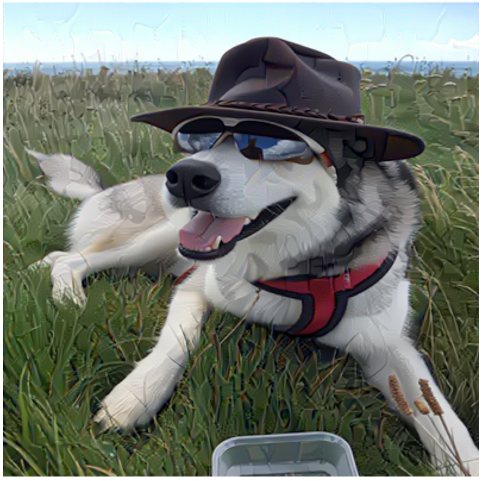
```
content_loss = 0.0
style_loss = 0.0

for f1, f2, f3 in zip(target_features, content_features, style_features):
    pass
    #计算 content_loss
    ######################################
    # write your code
    content_loss += torch.mean((f1 - f2)**2)
    ######################################

    #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)

    _ , b, c, d = f1.size()
    f1 = f1.view(b, c * d)
    f3 = f3.view(b, c * d)
    f1 = torch.mm(f1, f1.t())
    f3 = torch.mm(f3, f3.t())
    #gf1 = GM(f1)
    #gf3 = GM(f3)
    ######################################


    #计算 style_loss
    ######################################
    # write your code
    style_loss += torch.mean((f1 - f3)**2) / (b * c * d)
    ######################################


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

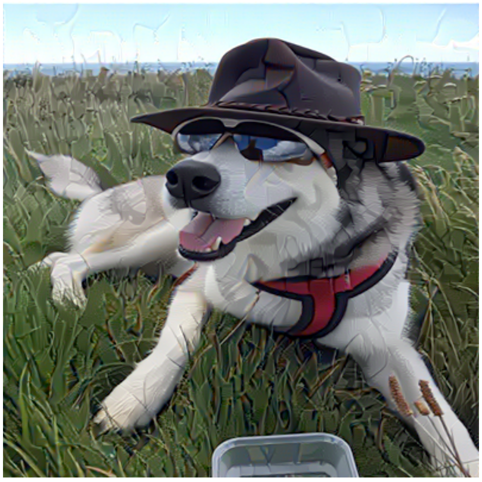
```
        print(content_loss.data)
        print(loss.data)
        ######################################

        #反向求导与优化
        ######################################
        # write your code
        optimizer.zero_grad()
        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=500)
    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=100)
    parser.add_argument('--lr', type=float, default=0.003)
    config = parser.parse_args()
    print(config)
    main(config)
```

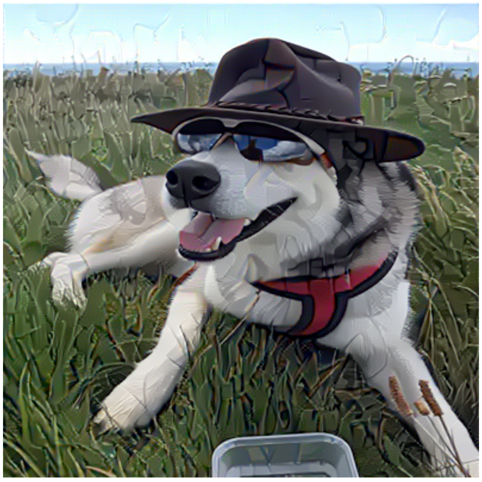
结论分析与体会：

将迭代次数改为 200

Step [10/200], Content Loss: 3.6908, Style Loss: 1019.4890
Step [20/200], Content Loss: 9.9866, Style Loss: 873.9231
Step [30/200], Content Loss: 14.3288, Style Loss: 778.0105
Step [40/200], Content Loss: 17.3717, Style Loss: 710.2642
Step [50/200], Content Loss: 19.6068, Style Loss: 658.8181
Step [60/200], Content Loss: 21.3440, Style Loss: 617.2078
Step [70/200], Content Loss: 22.7909, Style Loss: 582.3486
Step [80/200], Content Loss: 24.0091, Style Loss: 552.2748
Step [90/200], Content Loss: 25.0371, Style Loss: 525.8422
Step [100/200], Content Loss: 25.9181, Style Loss: 502.1700
Step [110/200], Content Loss: 26.7108, Style Loss: 480.6903
Step [120/200], Content Loss: 27.4222, Style Loss: 461.0207
Step [130/200], Content Loss: 28.0791, Style Loss: 442.9164
Step [140/500], Content Loss: 28.6689, Style Loss: 426.0802
Step [150/200], Content Loss: 29.2125, Style Loss: 410.3044
Step [160/200], Content Loss: 29.7101, Style Loss: 395.4733
Step [170/200], Content Loss: 30.1630, Style Loss: 381.4982
Step [180/200], Content Loss: 30.5872, Style Loss: 368.3009
Step [190/200], Content Loss: 30.9833, Style Loss: 355.8184
Step [200/200], Content Loss: 31.3585, Style Loss: 343.9487

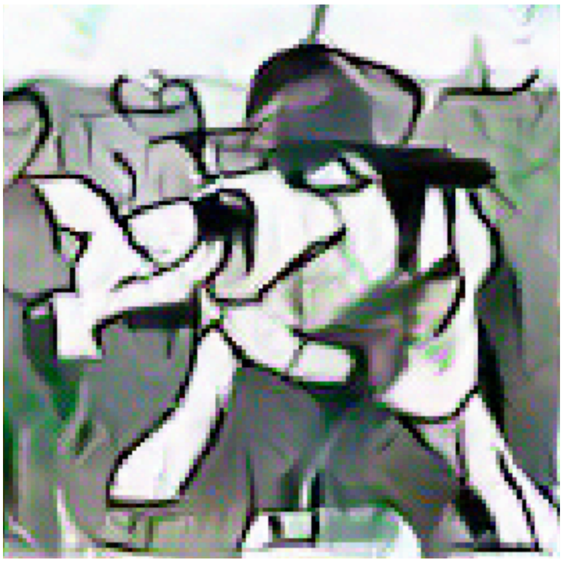




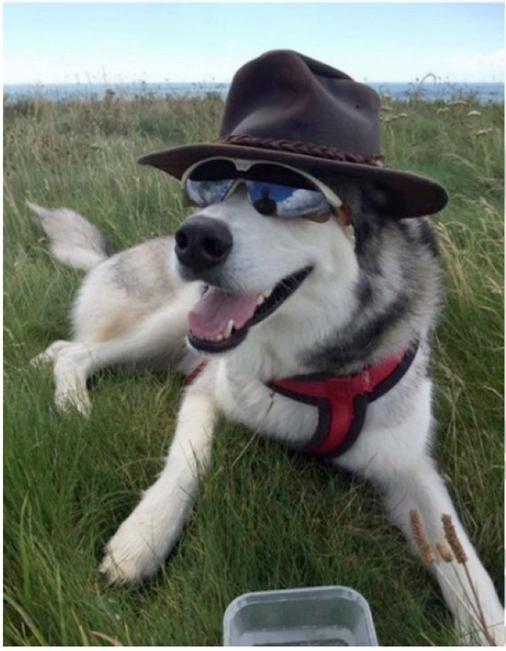
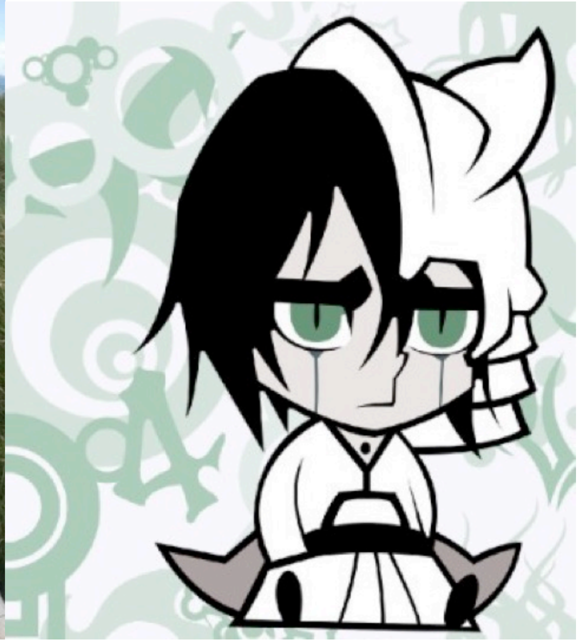


可以看到运行缓慢，但有一定效果

将 weight 改为 1000，优化器改为 LBFGS，将图片 resize，速度较快
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
3. 除大小的时候先平方后除