神经风格转换大作业报告

一、实验目的：神经风格转换（Neural Style Transfer）在深度学习领域是一个很有意思的算法，它能将风格图片的风格转换到内容图片上以生成一张新的图片。在本实验中，通过简单的代码（基于在线图像优化的慢速图像风格化迁移算法）实现神经风格转换，将风格图片的风格转换到内容图片上，得到一张新的图片，并输出迭代过程中的半成品。初步感受计算机视觉方面的深度学习。

二、设计思路

2.1 实验原理

在本实验中，首先通过 VGG 的高层特征表达接近内容图的特征表达（即内容相似），并用 Gram 矩阵来对图像中的风格进行建模和提取，利用慢速图像重建方法，让重建后的图像以梯度下降的方式更新像素值，使其 Gram 矩阵接近风格图的 Gram 矩阵（即风格相似），最终重建出来的结果图就既拥有风格图的风格，又有内容图的内容。那么，如何判定其内容损失，风格损失呢？即以什么为依据进行优化呢？

1、将内容图片简写为 C，风格图片简写为 S，生成图片简写为 G。那么内容损失函数定义为 $J(C,G)$，风格损失函数定义为 $J(S,G)$，总的损失函数为

$$L_{\text{total}} = \alpha L_{\text{content}} + \beta L_{\text{style}}$$

i）内容损失函数的计算：用 feature map 来计算内容损失函数，衡量内容的相似度。

$$L_{\text{content}} = \sum \| F^{l} - P^{l} \|^2$$

F^{l} 、P^{l} 为内容图片 C 和生成图片 G 在 CNN 的第 l 层经过激活函数后的值。所以当 F^{l} 与 P^{l} 越相似的时候，两图片的内容越相似。每层的 loss 可以设定一个权重得到总的内容损失，本次实验直接求取了平均值。

ii）风格损失函数的计算：我们使用 GRAM 矩阵来表示风格，对于提取出的特征，Gram 计算的实际上是两两特征之间的相关性，哪两个特征是同时出现的，哪两个是此消彼长的等等。因此，Gram 有助于把握整个图像的大体风格。因此要度量两个图像风格的差异，只需比较他们 Gram 矩阵的差异即可。

$$L_{\text{style}} = \frac{\sum_{\text{i=1}}^{C}\sum_{\text{j=1}}^{C}(G_{\text{ij}}^{l} - A_{\text{ij}}^{l})^2}{c * h * w}$$

G^{l} 与 A^{l} 是内容图片 C 和风格图片 S 在 l 层的 Gram 矩阵。c，h，w 为 l 层的高、宽、通道数。在多个层上分别比较他们的 Gram 矩阵的差异，每层的 loss 可以设定一个权重得到总的风格损失，本次实验直接求取了平均值。

2、利用 pytorch 实现加载一个特定的网络的张量，并且输出特定层的结果，根据这些中间层的输出结果，进行梯度回传，更新目标和参数，得到优化，非常方便。

如图 1 所示为神经风格迁移原理图。

2.2 流程设计

使用已经训练好的 VGG 模型，对内容图片、风格图片、目标图片逐层进行特征提取，并根据公式进行损耗计算。对损耗进行反向传播优化，更新目标图像以最小化内容图像的特征图与其特征图之间的均方误差以及样式图像的 Gram 矩阵与目标图像的 Gram 矩阵之间的均方误差以生成与样式图像的样式匹配的纹理。其最终目标是最小化总损耗，最大程度上保留内容，又融入了充足指定风格。图 2 是代码编写的主要流程图。

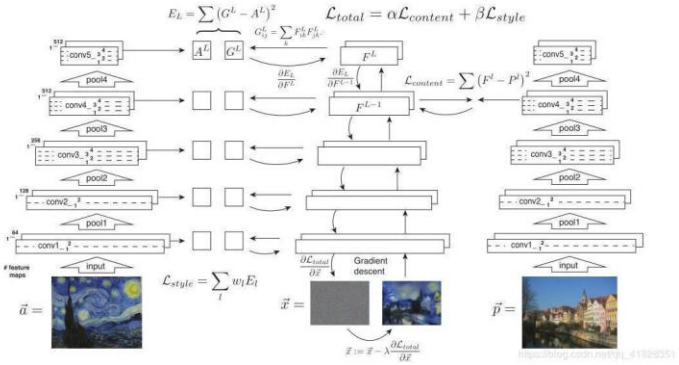

图 1 神经风格迁移原理图

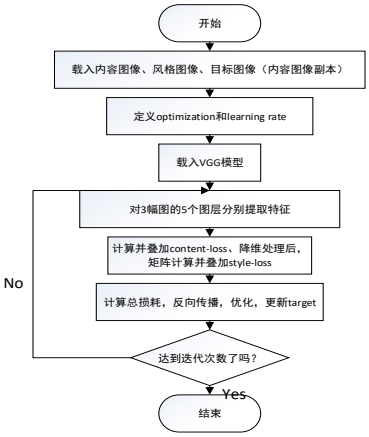

图 2 程序设计流程图

三、实验过程

代码来源：https://github.com/yunjey/pytorch-tutorial/tree/master/tutorials/03-advanced/neural_style_transfer

3.1 实验环境

Python3.7、argparse、torch、torchvision、Pillow

3.2 关键代码

1、使用网络中的 5 个卷积层，进行特征提取。

```
class VGGNet(nn.Module): #继承nn.module类,明确告诉它应该更新模型的哪些参数(张量)
    def __init__(self):
        """Select conv1_1 ~ conv5_1 activation maps.选择隐藏层、激活函数"""
        super(VGGNet, self).__init__()
        self.select = ['0', '5', '10', '19', '28']   #标号即层号，我们要保存的是['0', '5', '10', '19', '28'] 的输出结果。
        self.vgg = models.vgg19(pretrained=True).features

    def forward(self, x):
        """Extract multiple convolutional feature maps.提取多个卷积特征图"""
        features = []
        for name, layer in self.vgg._modules.items():
            x = layer(x)
            if name in self.select:
                features.append(x)
        return features
```

2、设置优化参数为目标图像 target，当 loss 被反向传播进行优化时，target 会被不断更新。

```
    optimizer = torch.optim.Adam([target], lr=config.lr, betas=[0.5, 0.999])  #优化，[待优化参数]，【学习率】，【系数】
    vgg = VGGNet().to(device).eval()
```

3、对每幅图进行特征提取，计算内容损耗，将特征 reshape 成二维 gram 矩阵后进行计算得到风格损失

```
#迭代优化
for step in range(config.total_step):

    # Extract multiple(5) conv feature vectors   提取5个特征向量（5层）
    target_features = vgg(target)
    content_features = vgg(content)
    style_features = vgg(style)

    style_loss = 0
    content_loss = 0
    for f1, f2, f3 in zip(target_features, content_features, style_features):
        # Compute content loss with target and content images计算content loss
        content_loss += torch.mean((f1 - f2) ** 2)

        # Reshape convolutional feature maps 将特征reshape成二维矩阵
        _, c, h, w = f1.size()
        f1 = f1.view(c, h * w)
        f3 = f3.view(c, h * w)

        # Compute gram matrix  gram矩阵
        f1 = torch.mm(f1, f1.t())
        f3 = torch.mm(f3, f3.t())

        # Compute style loss with target and style images
        style_loss += torch.mean((f1 - f3) ** 2) / (c * h * w)
```

3、计算总损失，梯度回传，进行迭代优化，更新目标图像。

```
        # Compute total loss, backprop and optimize
    loss = content_loss + config.style_weight * style_loss
    optimizer.zero_grad()
    loss.backward()   #loss回传，反向求导，梯度下降
    optimizer.step()  #更新参量
```

4、参数传入：在这里，传入迭代次数、风格比重等参数。我们设定迭代次数为 2000 次，每隔 10 次输出 loss 情况，每隔 100 次输出一次重建图像（源代码为 500 次），以更好的感受图像的变化。

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('--content', type=str, default='C:\\Users\\cwt\\PycharmProjects\\BT\\text\\png\\content.jpg')
    parser.add_argument('--style', type=str, default='C:\\Users\\cwt\\PycharmProjects\\BT\\text\\png\\style.png')
    parser.add_argument('--max_size', type=int, default=400)
    parser.add_argument('--total_step', type=int, default=2000)
    parser.add_argument('--log_step', type=int, default=10)
    parser.add_argument('--sample_step', type=int, default=100)
    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)
```

三、实验结果

1.图 3 是目标图像逐步优化的过程，可以看到内容图片的细节正在慢慢损失，但保留了主要特征，并在优化过程中不断地融入目标风格。

2.图 4 是使用同一张图，融入不通过风格图片的结果。每种风格都很好地融入了图像。

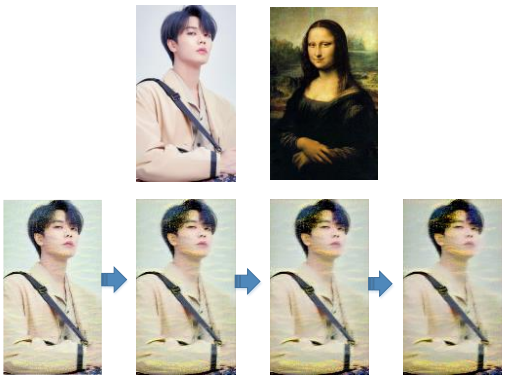

图 3 目标优化

图 4 不同风格

四、实验感想

神经风格转移定义两个损失函数，让网络的输出在这两个 loss 之间做一个权衡，是 style 与 content 的互补关系。作者用的是一个训练好的网络，与传统的自己搭建网络不同，优化的目标不是网络的参数，而是一个 target，这个 target 是网络的输入，是一个包含梯度信息的变量。使用 pytorch 实现进行梯度回传，进行图像重建，也非常方便实现。本次的实验代码结构较为简单，让我这个初学者初步接触到了机器学习，实验的内容也非常有趣，但其中还是有许多需要深入学习的东西。

此外，我还录制了一个简短的视频，简单对本次大作业进行了介绍和总结，方便初次接触机器学习的同学进行学习。

视频地址：https://www.bilibili.com/video/BV1L5411x7W1/