



# 深度学习导论

## 实验 5

姓名 徐海阳 学号 PB20000326 院系 少年班学院

### 摘要

本实验的内容为风格迁移（Style Transfer）。首先，阐述风格迁移开山之作”Image Style Transfer Using Convolutional Neural Networks”的基本原理；然后，使用 PyTorch 神经网络库复现了这篇论文的效果；最后，在大量的图片对间实现风格迁移，效果很棒。

### 目录

<b>1 基本原理与方法</b>	<b>2</b>
1.1 原理	2
1.2 方法	3
<b>2 实践过程及关键代码</b>	<b>4</b>
<b>3 实验结果</b>	<b>6</b>

# 1 基本原理与方法

## 1.1 原理

每一幅画，都可以看成「内容」与「风格」的组合。这篇文章通过使用预训练的 CNN 成功地将图像中的内容和风格分离了出来，并在风格迁移任务上实现了很好的感知效果。所谓风格迁移，就是把一张图片的风格，嵌入到另一张图片的内容里，形成一张新的图片。如下图1所示，B、C、D 是 A 的内容与三种不同风格组合形成的图片。

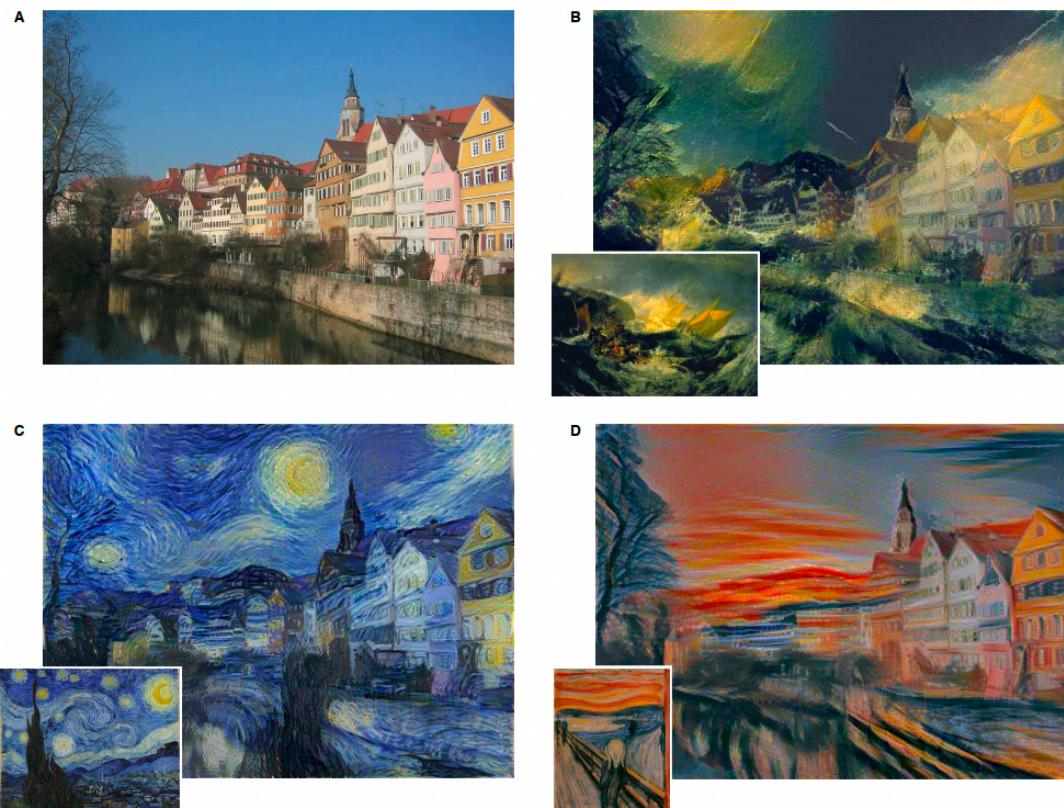


图 1: 风格迁移示例

那么，如何实现风格迁移呢？

这篇文章的作者发现，如果用预训练 VGG 模型不同层的卷积输出作为拟合特征，则可以拟合出不同的图像，如图2所示：

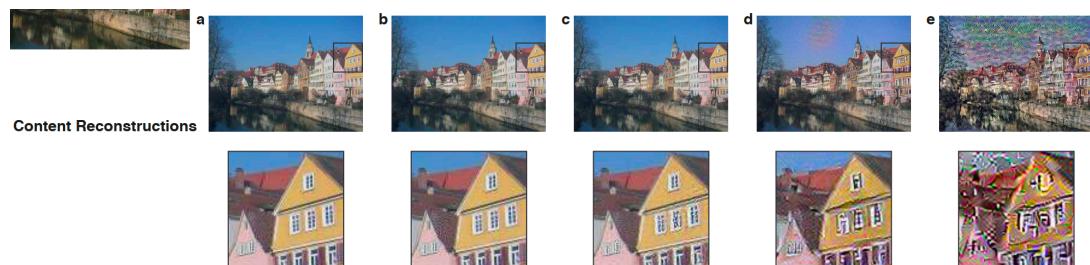


图 2: 拟合内容

上图中，越靠右边的图像，是用越深的卷积层特征进行特征拟合恢复出来的图像。从这些图像

恢复结果可以看出，更深的特征只会保留图像的内容（形状），而难以保留图像的纹理（天空的颜色、房子的颜色）。那么，这些图片具体是怎么拟合出来的呢？让我们和刚刚一样，详细地看一看这一图像生成过程。假设我们想生成上面的图 c，即第三个卷积层的拟合结果。我们已经得到了模型 model\_conv123，其包含了预训练 VGG 里的前三个卷积层。我们可以设立以下的优化目标：目标特征为目标图片通过这三层卷积层后得到的特征，然后初始化一个随机的源图像，也通过这三层卷积层后得到源特征。定义损失为源特征和目标特征之间的 L2 损失，然后进行优化（梯度下降即可）。优化的结果应该能使得源图像与目标图像很接近。

同时，他们发现不仅是卷积结果可以当作拟合特征，VGG 的一些其他中间结果也可以作为拟合特征。受到之前用 CNN 做纹理生成的工作的启发，他们发现用卷积结果的 Gram 矩阵作为拟合特征可以得到另一种图像生成效果，如图3所示：

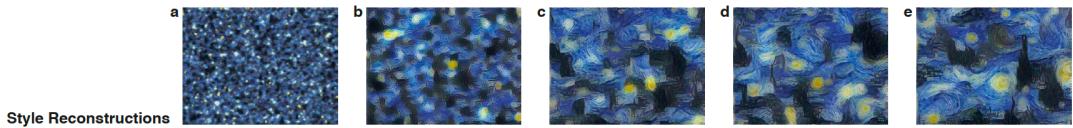


图 3: 拟合风格

上图中，右边 a-e 是用 VGG 不同卷积结果的 Gram 矩阵作为拟合特征，得到的对左图的拟合图像。可以看出，用这种特征来拟合的话，生成图像会失去原图的内容（比如星星和物体的位置完全变了），但是会保持图像的整体风格。这里稍微提一下 Gram 矩阵的计算方法。Gram 矩阵定义在两个特征的矩阵  $F_1, F_2$  上。其中，每个特征矩阵  $F$  是 VGG 某层的卷积输出张量  $F_{conv}(shape: [n, h, w])$  reshape 成一个矩阵  $F$  ( $shape: [n, h * w]$ ) 的结果。Gram 矩阵，就是两个特征矩阵  $F_1, F_2$  的内积，即  $F_1$  每个通道的特征向量和  $F_2$  每个通道的特征向量的相似度构成的矩阵。我们这里假设  $F_1=F_2$ ，即对某个卷积特征自身生成 Gram 矩阵。Gram 矩阵表示的是通道之间的相似性，与位置无关。因此，Gram 矩阵是一种具有空间不变性（spatial invariance）的指标，可以描述整幅图像的性质，适用于拟合风格。与之相对，我们之前拟合图像内容时用的是图像每一个位置的特征，这一个指标是和空间相关的。

如此一来，我们知道了如何拟合内容，这一节知道了怎么去拟合风格。要把二者结合起来，只要令我们的优化目标既包含和内容图像的内容误差，又包含和风格图像的风格误差即可。

## 1.2 方法

首先，我们需要一个预训练的卷积神经网络（CNN），例如 VGG-19，来提取图像的内容和风格特征。CNN 由多个卷积层和池化层组成，每一层都可以看作是一个图像滤波器，从低层到高层提取不同层次的图像信息。

然后，我们定义一个内容损失函数，用来衡量两幅图像在某一层的 CNN 特征之间的差异。一般来说，我们选择一个较高层的 CNN 特征作为内容表示，例如第四个卷积块的第二个卷积层 (conv4\_2)。内容损失函数可以表示为：

$$L_{content}(p, x, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2$$

其中， $p$  是内容图像， $x$  是生成图像， $l$  是 CNN 的某一层， $F'$  和  $P'$  分别是  $x$  和  $p$  在该层的特征映射， $i$  和  $j$  是特征映射的索引。

接着，我们定义一个风格损失函数，用来衡量两幅图像在所有层的 CNN 特征之间的风格差异。风格差异可以用两幅图像在每一层的特征映射之间的格拉姆矩阵（Gram matrix）来度量。格拉姆矩

阵是特征映射的内积矩阵, 反映了不同特征之间的相关性。风格损失函数可以表示为:

$$L_{\text{style}}(a, x) = \sum_{l=0}^L w_l E_l$$

其中,  $a$  是风格图像,  $x$  是生成图像,  $L$  是 CNN 的总层数,  $w_l$  是第  $l$  层的权重系数,  $E_l$  是第  $l$  层的风格误差, 定义为:

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2$$

其中,  $G^l$  和  $A^l$  分别是  $x$  和  $a$  在第  $l$  层的格拉姆矩阵,  $N_l$  是第  $l$  层的特征数量,  $M_l$  是第  $l$  层的特征维度。

最后, 我们定义一个总损失函数, 用来同时优化内容和风格的匹配程度。总损失函数是内容损失函数和风格损失函数的加权和, 可以表示为:

$$L_{\text{total}}(p, a, x) = \alpha L_{\text{content}}(p, x, l) + \beta L_{\text{style}}(a, x)$$

其中,  $\alpha$  和  $\beta$  是内容和风格的权重系数。我们使用梯度下降法来最小化总损失函数, 并更新生成图像  $x$ 。最终得到的生成图像  $x^*$  既保留了内容图像  $p$  的内容信息, 又融合了风格图像  $a$  的风格信息。

## 2 实践过程及关键代码

定义如图4内容损失函数:

$$L_{\text{content}}(p, x, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2$$

```

63 class ContentLoss(nn.Module):
64     def __init__(self, target, weight):
65         super(ContentLoss, self).__init__()
66         self.target = target.detach() * weight
67         self.weight = weight
68         self.criterion = nn.MSELoss()
69
70     def forward(self, input):
71         output = input.clone()
72         content = input
73         content *= self.weight
74         self.loss = self.criterion(content, self.target)
75         return output
76
77     def backward(self, retain_graph=True):
78         self.loss.backward(retain_graph=retain_graph)
79         return self.loss

```

图 4: Content Loss

定义如图5 Grammer 函数:

定义如图6风格损失函数:

$$L_{\text{style}}(a, x) = \sum_{l=0}^L w_l E_l$$

训练的前向过程如图7:

利用 LBFGS 进行优化如图8:

```
81 class GramMatrix(nn.Module):
82
83     def forward(self, input):
84
85         B, C, H, W = input.shape
86         features = input.reshape(B * C, H * W)
87
88         # compute gram matrix G and normalize
89         G = features @ features.T / (B * C * H * W)
90
91         return G
```

图 5: Gram Matrix

```
93 class StyleLoss(nn.Module):
94
95     def __init__(self, target, weight):
96         super(StyleLoss, self).__init__()
97         self.target = target.detach() * weight
98         self.weight = weight
99         self.gram = GramMatrix()
100        self.criterion = nn.MSELoss()
101
102    def forward(self, input):
103        output = input.clone()
104        style = self.gram(input)
105        style *= self.weight
106        self.loss = self.criterion(style, self.target)
107        return output
108
109    def backward(self, retain_graph=True):
110        self.loss.backward(retain_graph=retain_graph)
111        return self.loss
```

图 6: Style Loss

```
29     cnn = copy.deepcopy(cnn)
30     for layer in list(cnn):
31         if isinstance(layer, nn.Conv2d):
32             name = "conv_" + str(i)
33             model.add_module(name, layer)
34             if name in content_layers:
35                 # add content loss:
36                 target = model(content_img).clone()
37                 content_loss = ContentLoss(target, content_weight)
38                 model.add_module("content_loss_" + str(i), content_loss)
39                 content_losses.append(content_loss)
40             if name in style_layers:
41                 # add style loss:
42                 target_feature = model(style_img).clone()
43                 target_feature_gram = gram(target_feature)
44                 style_loss = StyleLoss(target_feature_gram, style_weight)
45                 model.add_module("style_loss_" + str(i), style_loss)
46                 style_losses.append(style_loss)
47             if isinstance(layer, nn.ReLU):
```

图 7: Forward

```
73     input_param = nn.Parameter(input_img.data)
74     optimizer = optim.LBFGS([input_param])
75
76     run = [0]
77     while run[0] <= args.epoch:
78
79         def closure():
80             input_param.data.clamp_(0, 1)
81
82             optimizer.zero_grad()
83             model(input_param)
84             style_loss = 0
85             content_loss = 0
86
87             for sl in style_losses:
88                 style_loss += sl.backward()
89             for cl in content_losses:
90                 content_loss += cl.backward()
91
92             run[0] += 1
93             if run[0] % 50 == 0:
94                 print(f'{run[0]}/{args.epoch}:')
95                 print('Style Loss: {:.4f} Content Loss: {:.4f}'.format\
96                     (style_loss.item(), content_loss.item()))
97
98             loss = style_loss + content_loss
99             return loss
100
101     optimizer.step(closure)
```

图 8: Step

### 3 实验结果

实验结果如下图所示，左边为原图，右边为六种风格生成的图片。



图 9: 金门大桥



图 10: 帝国大厦



图 11: 大本钟



图 12: 金字塔



图 13: 伦敦塔桥



图 14: 卢浮宫

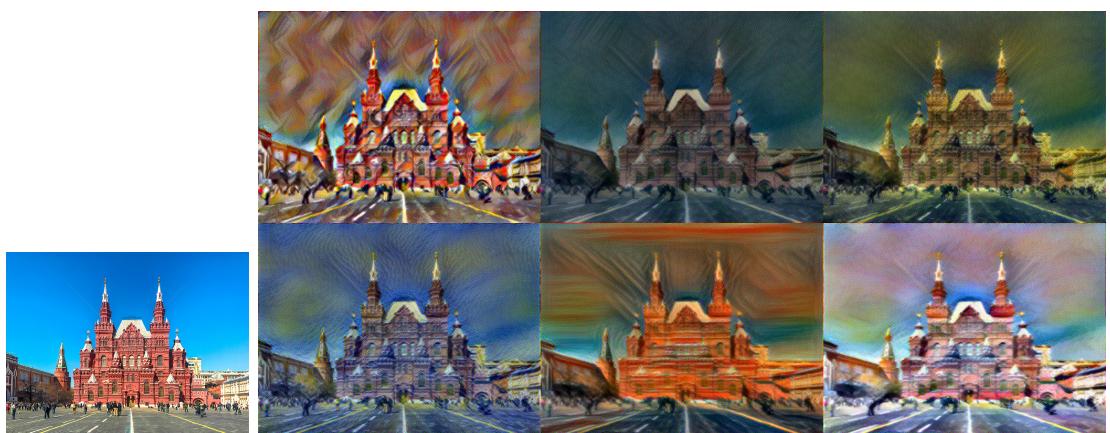


图 15: 红场