摘要 图像风格迁移是计算机视觉中兼具科技理性与艺术美感的一项技术，其中 2015 年 Gatys 等人的工作尤为重要。本文基于 Gatys 等人的工作，通过计算内容损失和风格损失构建画家梵高的风格迁移图像。并且，基于 CS224n 课程的技术刻画了 TV 损失，使得整个风格迁移结果更为平顺。通过数值计算、梯度下降，我们使梵高的画笔再一次灵动于纸上，跨时空为《原神》勾勒出新的轮廓。

关键字：风格迁移、图像处理、计算机视觉、原神

一、问题背景

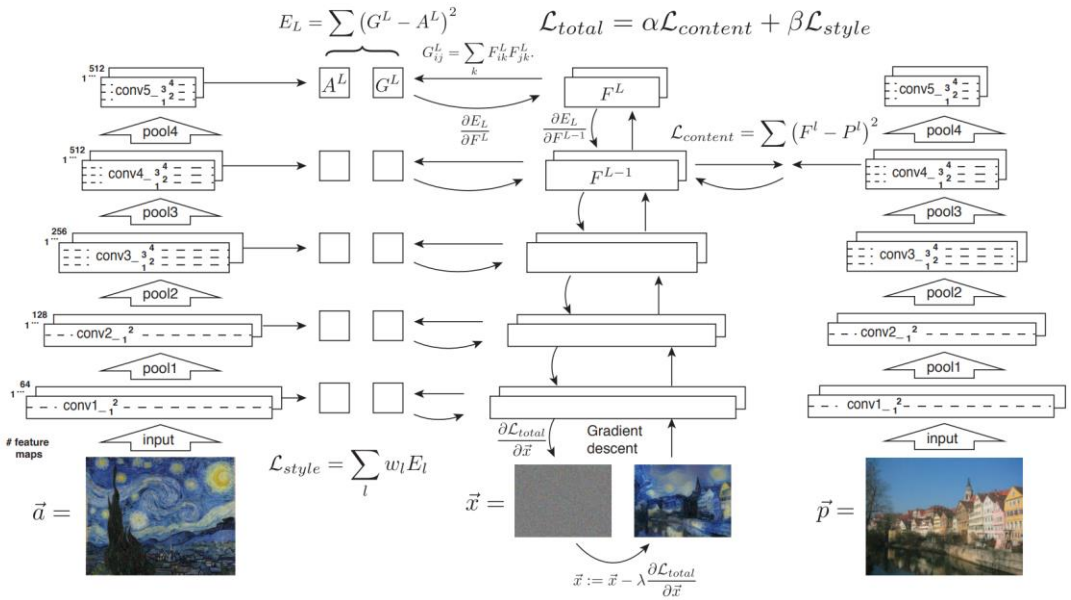
图像风格迁移是计算机视觉领域中的一项技术，它允许将一张图片的艺术风格迁移到另一张图片上，同时保留内容图像的结构。这项技术的基础在于神经网络，特别是 Gatys 等人[1]在 2016 年 CVPR 论文 *Image Style Transfer Using Convolutional Neural Networks* 中提出的算法。此外，整个风格迁移的训练过程也是解决一个最优化问题，我们使用梯度下降的方法，并在优化器中使用了模拟退火的技巧来解决这个问题。本文参考上述这篇文献复现并进行一些小小改进。

图像风格迁移的目标是创建一个新图像，该图像结合了给定内容图像的语义信息和给定风格图像的外观特征。传统的图像处理方法难以实现这一点，因为它们缺乏能够显式表示语义信息的图像表示形式，因此无法有效地分离内容与风格。Gatys 等人的研究利用了为物体识别优化的 CNN，这些网络可以捕捉到高级别的图像信息，从而实现了内容与风格的分离及重组。

二、所用算法介绍

Gatys 等人提出了一个基于 CNN 的算法，该算法通过纹理合成的方法，在单个神经网络内将纹理模型转换为一个优化问题。具体来说，他们使用了一个预训练的 VGG 网络，其中不包括全连接层，并对卷积层进行了归一化处理。为了生成新的图像，他们执行了反图像搜索以匹配示例图像的特征表示。对于图像合成，作者发现用平均池化代替最大池化可以获得更吸引人的结果。

在数学上，总的损失函数由内容损失和风格损失组成，如下所示：



1. 内容和风格特征提取

[1] https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Gatys_Image_Style_Transfer_CVPR_2016_paper.pdf

图像风格迁移算法首先需要提取内容图像和风格图像的特征。这些特征通过预训练的卷积神经网络（如 VGG 网络）提取。

内容图像（右下角）和风格图像（左下角）分别通过预训练的卷积神经网络提取特征。- 内容图像的特征表示为 P1，风格图像的特征表示为 AL，其中 L，1 表示网络的层。

2. 风格损失计算

风格损失通过计算生成图像的格拉姆矩阵与风格图像的格拉姆矩阵之间的差异来计算。

格拉姆矩阵 G 通过以下公式计算：

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$$

其中 FL 是生成图像在第 L 层的特征。

风格损失通过计算生成图像的格拉姆矩阵 G 与风格图像的格拉姆矩阵 A 之间的差异来计算：

$$E_l = \sum (G^l - A^l)^2$$

$$\mathcal{L}_{style} = \sum_l w_l E_l$$

3. 内容损失计算

内容损失通过计算生成图像的特征与内容图像的特征之间的差异来计算。

$$\mathcal{L}_{content} = \sum (F^l - P^l)^2$$

4. 总损失计算

总损失是内容损失和风格损失的线性组合。

$$\mathcal{L}_{total} = \alpha \mathcal{L}_{content} + \beta \mathcal{L}_{style}$$

5. 优化过程

通过梯度下降法最小化总损失，更新生成图像的像素值。

$$\bar{x} := \bar{x} - \lambda \frac{\partial \mathcal{L}_{total}}{\partial \bar{x}}$$

更进一步，我们加入 Total Various Loss，使得生成的风格迁移图像更加平滑

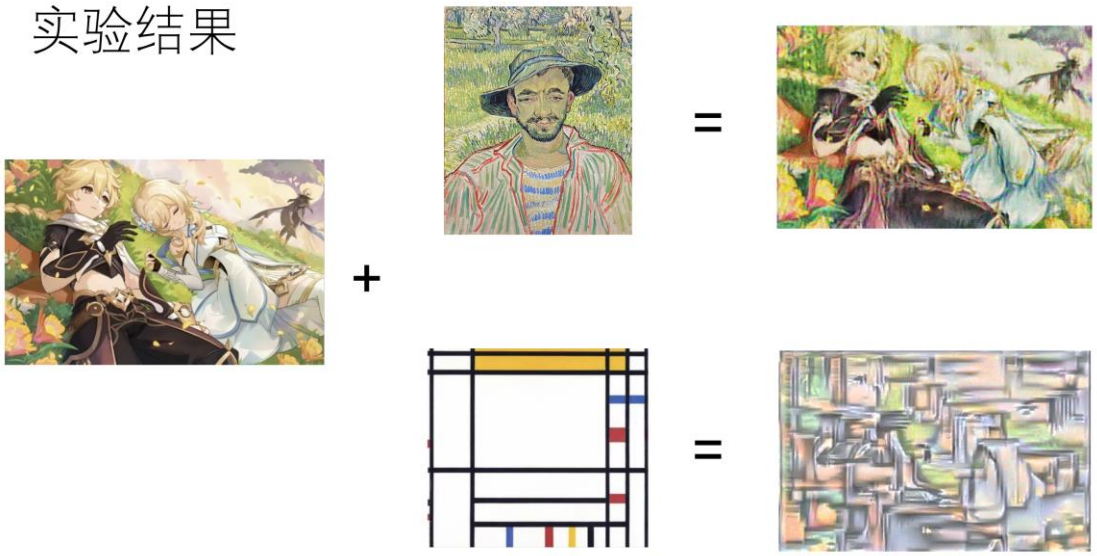
$L_{tv} = w_t \times \left(\sum_{c=1}^{3} \sum_{i=1}^{H-1} \sum_{j=1}^{W} (x_{i+1,j,c} - x_{i,j,c})^2 + \sum_{c=1}^{3} \sum_{i=1}^{H} \sum_{j=1}^{W-1} (x_{i,j+1,c} - x_{i,j,c})^2 \right)$

所以最终的 Loss 函数：

$$\mathcal{L}_{total} = \alpha \mathcal{L}_{content} + \beta \mathcal{L}_{style} + \gamma \mathcal{L}_{total\_various}$$

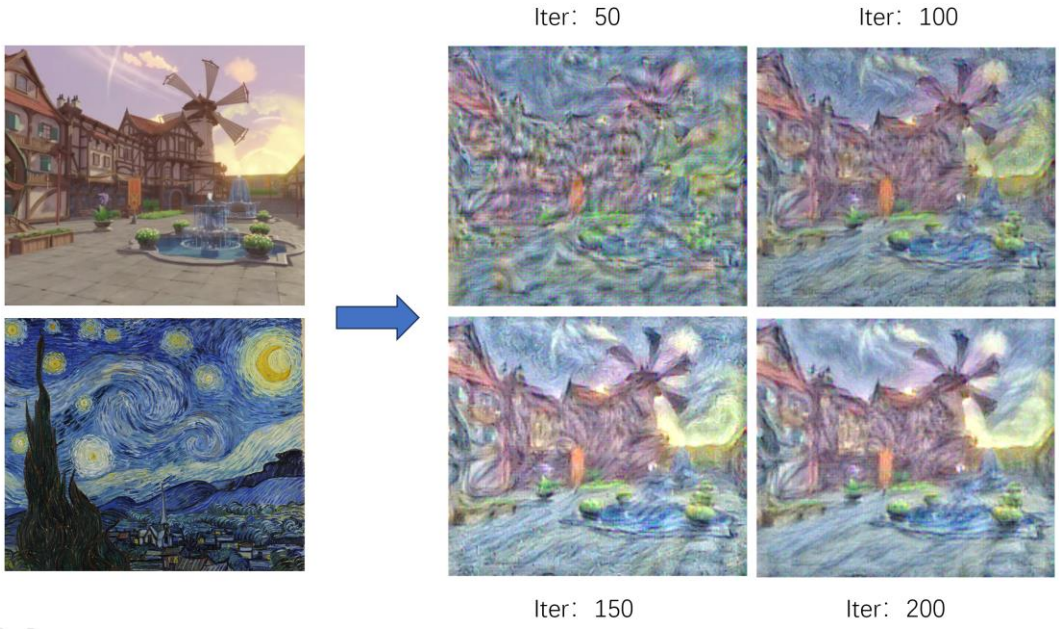
三、结果展示和分析

如下图所示，在加入了梵高（印象派画家）的《园丁》和蒙德里安（抽象派画家）的《红黄蓝》作品各自风格后，原神的图片得以有不同的结果展现：

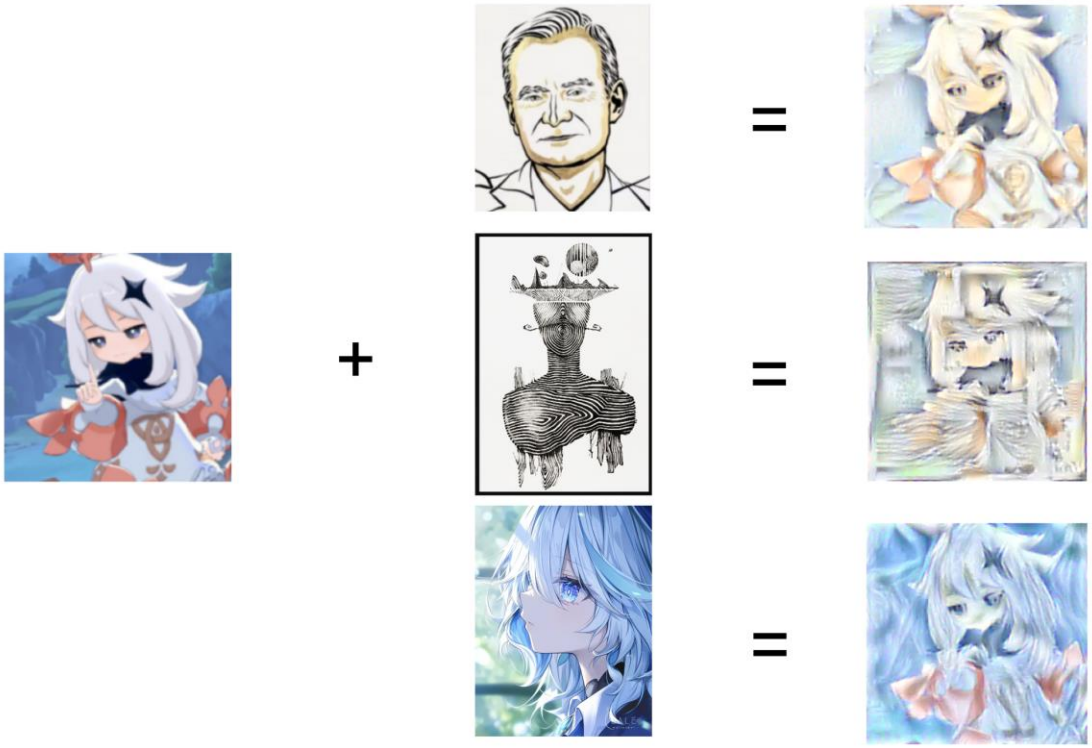


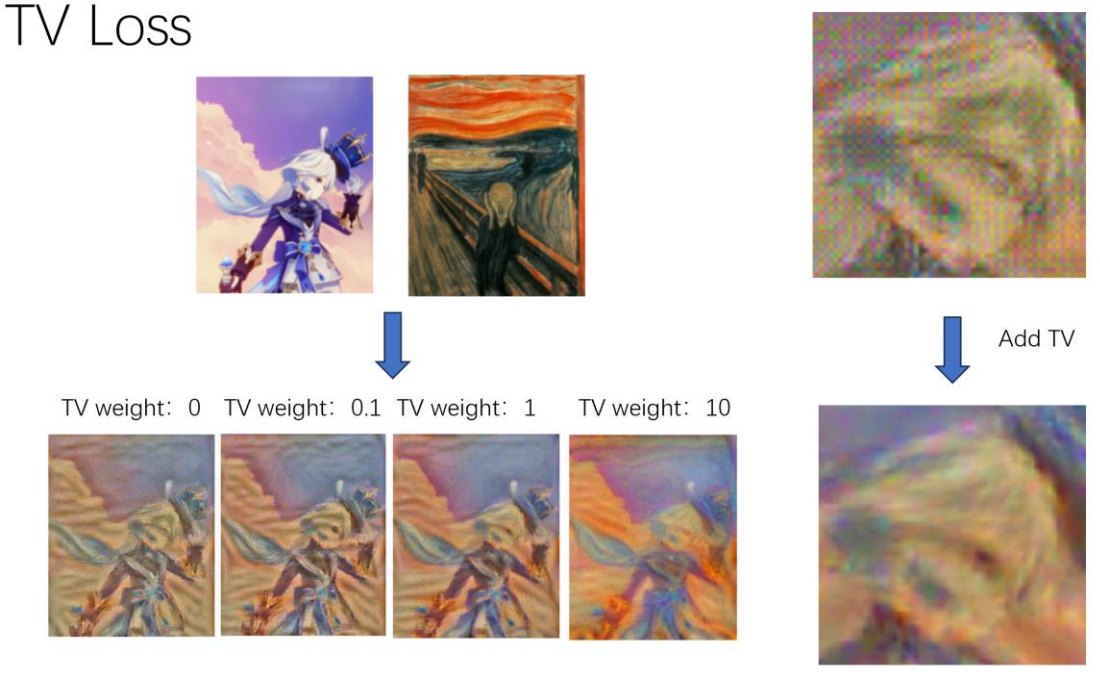
我们可以看到对于抽象化的“横平竖直”的方块化结构，我们可以成功提取其特征。而迁移风格梵高的画风后，我们得到的图像有更多“浮世绘”一般的笔触（也就是短线条），可以看到我们成功迁移了这些画的风格。

而对于原神的一些街景，我们用梵高的《星夜空》做风格迁移，也可以处理出类似的结果：



图中展示了不同训练步数下的结果，在 150-200 循环时取得最好的结果，我们可以发现，通过不断地迭代，风格信息被更优的加入，从一开始的散乱条纹逐渐变成更为有组织的区域条纹，勾勒出房子、风车的轮廓，更多例子展现如下。





总的来说，我们加入的 TV 损失使得图片结果更加平滑，而我们也成功迁移了不同梵高作品，不同其他图像的风格，来生成对应图片，也就实现了跨越时空的艺术创作“让梵高画出原神游戏中的景观”。

四、代码说明

1. 北太天元代码

我们可以用北太天元做基本上述内容损失、风格损失、TV 损失的计算，并整合到一起，具体代码在 example.m 文件中，下面为简单说明：

Gram矩阵

```
% 定义函数
function result = gram_matrix(features, normalize)
    [C, H, W] = size(features);
    first = reshape(features, [C, H*W]);
    result = first * first';
    if normalize
        result = result / (H*W*C);
    end
end
```

TV损失

```
function loss = tv_loss(img, tv_weight)
    up = img(:, 1:end-1, :);
    down = img(:, 2:end, :);
    left = img(:, :, 1:end-1);
    right = img(:, :, 2:end);
    first_sum = sum(sum((up - down).^2));
    second_sum = sum(sum((left - right).^2));
    loss = tv_weight * (sum(first_sum) + sum(second_sum));
end
```

风格损失

```
function style_losses = style_loss(feats, style_layers, style_targets, style_weights)
    style_losses = 0;
    for i = 1:length(style_layers)
        now = gram_matrix(feats(:, style_layers(i), :),1) - style_targets{i};
        style_losses = style_losses + style_weights(i) * sum(now(:).*now(:));
    end
end
```

内容损失

```
function loss = content_loss(content_weight, content_current, content_original)
    diffmat = content_current - content_original;
    loss = content_weight * sum(diffmat(:).*diffmat(:));
end
```

但是由于北太天元目前做梯度下降比较困难，并且矩阵转置不支持 3 维以上，所以我们后续训练使用 python 进行训练。但是核心计算函数北太天元都能实现。

2. python 代码

我们使用 pytorch 架构构建机器学习代码，详情见 StyleTransfer-core.ipynb，里面有较为详细的说明。

Gram矩阵

```
def gram_matrix(features, normalize=True):
    N,C,H,W=features.shape
    first=features.view([N,C,-1]) #合并H，W变为N，C，H*W
    second=first.permute([0,2,1])
    result=torch.matmul(first,second)
    if normalize:
      result=result/(H*W*C)

    return result
```

TV损失

```
def tv_loss(img, tv_weight):
    up=img[:,:,:-1,:]
    down=img[:,:,1:,:]
    first_sum=torch.sum((up-down)**2)
    left=img[:,:,:,:-1]
    right=img[:,:,:,1:]
    second_sum= torch.sum((left-right)**2)
    loss=tv_weight*(first_sum+second_sum)
    return loss
```

风格损失

```
def style_loss(feats, style_layers, style_targets, style_weights):
    style_losses=0
    for i in range(len(style_layers)):
        now=gram_matrix(feats[style_layers[i]])-style_targets[i]
        style_losses+=style_weights[i]*torch.sum(now*now)
    return style_losses
```

总损失

```
# Compute loss
c_loss = content_loss(content_weight, feats[content_layer], content_target)
s_loss = style_loss(feats, style_layers, style_targets, style_weights)
t_loss = tv_loss(img, tv_weight)
loss = c_loss + s_loss + t_loss
```

内容损失

```
def content_loss(content_weight, content_current, content_original):
    diffmat=content_current-content_original
    lost=diffmat*diffmat
    return content_weight*torch.sum(lost)
```

梯度下降

```
# Set up optimization hyperparameters
initial_lr = 3.0
decayed_lr = 0.1
decay_lr_at = 180

optimizer = torch.optim.Adam([img], lr=initial_lr)
```

五、总结

我们成功利用构造内容损失、风格损失和总方差损失（TV 损失）完成了著名印象派画家梵高的风格迁移。其中 TV 损失的加入可以为迁移图片提供更为平滑的刻画，减少生成的图片的颗粒度。

参考资料：

1. Image Style Transfer Using Convolutional Neural Networks" (Gatys et al., CVPR 2015)
2. 斯坦福 CS231n 课程资料：https://cs231n.github.io/
3. 原神官网：https://ys.mihoyo.com/