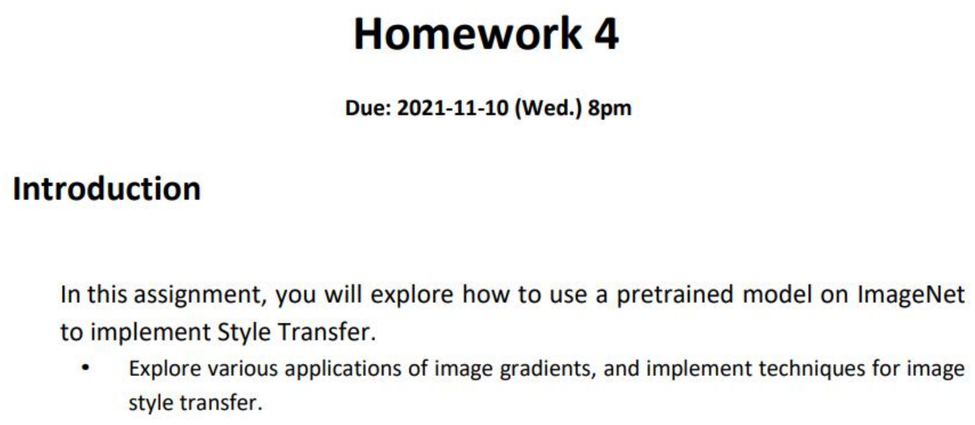
计算机科学与技术学院<u>神经网络与深度学习</u>课程实验报告

实验题目：Style Transfer		学号：201900301174
日期：2021.11.10	班级： 智能 19	姓名： 韩旭
Email：hanx@mail.sdu.edu.cn		

实验目的：



Homework 4

Due: 2021-11-10 (Wed.) 8pm

Introduction

In this assignment, you will explore how to use a pretrained model on ImageNet to implement Style Transfer.

- Explore various applications of image gradients, and implement techniques for image style transfer.

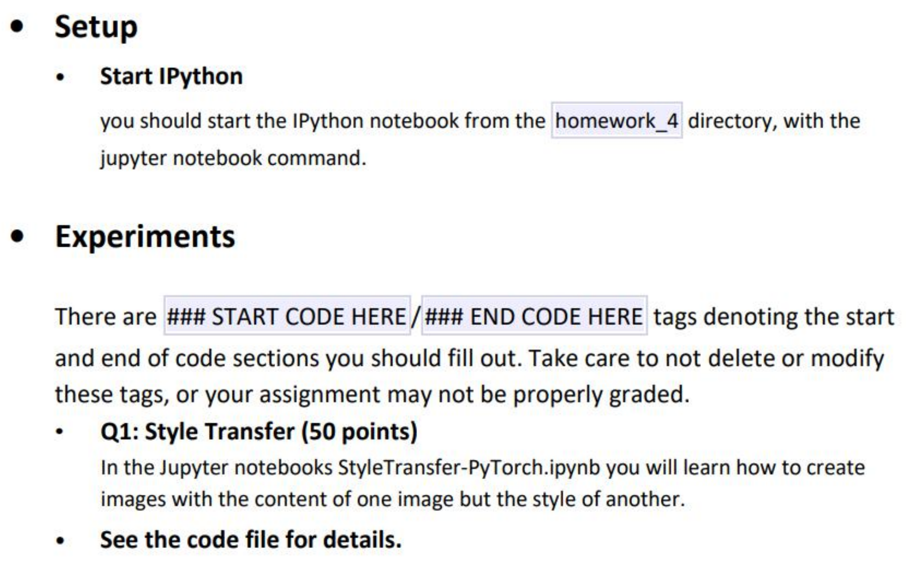
实验软件和硬件环境：

Jupyter notebook

Xeon gold 6226e

RTX3090

实验原理和方法：



- **Setup**
  - **Start IPython**

    you should start the IPython notebook from the `homework_4` directory, with the jupyter notebook command.

- **Experiments**

  There are `### START CODE HERE` / `### END CODE HERE` tags denoting the start and end of code sections you should fill out. Take care to not delete or modify these tags, or your assignment may not be properly graded.

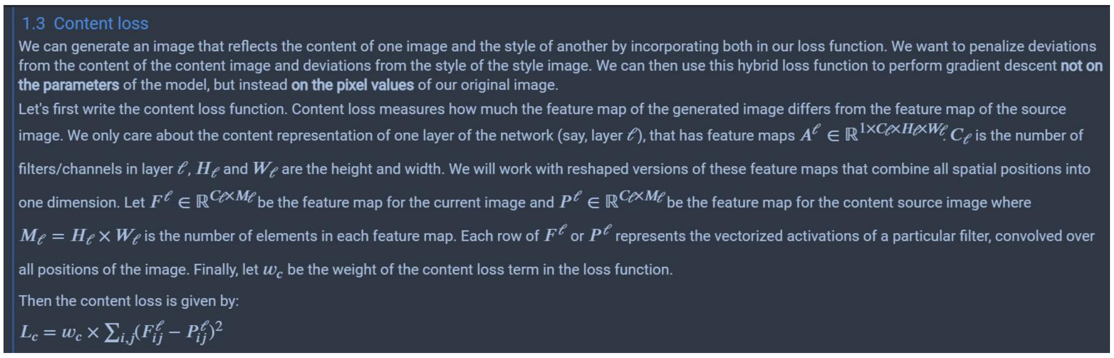
  - **Q1: Style Transfer (50 points)**

    In the Jupyter notebooks StyleTransfer-PyTorch.ipynb you will learn how to create images with the content of one image but the style of another.

  - **See the code file for details.**

实验步骤：（不要求罗列完整源代码）

1.content loss



1.3 Content loss

We can generate an image that reflects the content of one image and the style of another by incorporating both in our loss function. We want to penalize deviations from the content of the content image and deviations from the style of the style image. We can then use this hybrid loss function to perform gradient descent **not on the parameters** of the model, but instead **on the pixel values** of our original image.

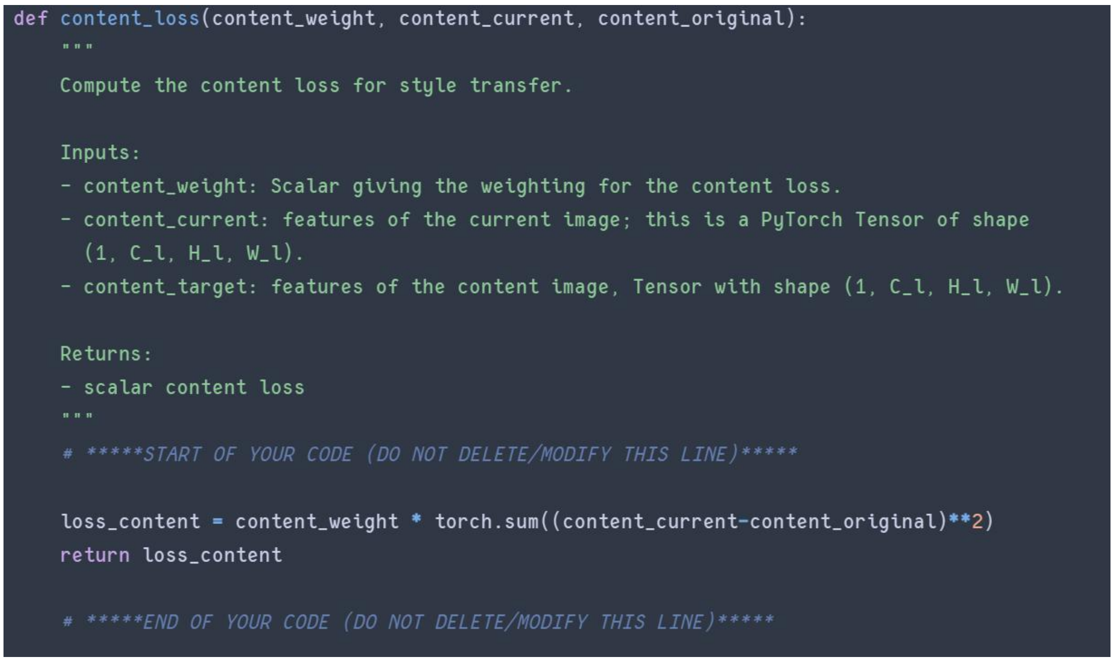
Let's first write the content loss function. Content loss measures how much the feature map of the generated image differs from the feature map of the source image. We only care about the content representation of one layer of the network (say, layer ℓ), that has feature maps $A^\ell \in \mathbb{R}^{1\times C_\ell \times H_\ell \times W_\ell}$. C_ℓ is the number of filters/channels in layer ℓ, H_ℓ and W_ℓ are the height and width. We will work with reshaped versions of these feature maps that combine all spatial positions into one dimension. Let $F^\ell \in \mathbb{R}^{C_\ell \times M_\ell}$ be the feature map for the current image and $P^\ell \in \mathbb{R}^{C_\ell \times M_\ell}$ be the feature map for the content source image where $M_\ell = H_\ell \times W_\ell$ is the number of elements in each feature map. Each row of F^ℓ or P^ℓ represents the vectorized activations of a particular filter, convolved over all positions of the image. Finally, let w_c be the weight of the content loss term in the loss function.

Then the content loss is given by:

$$L_c = w_c \times \sum_{i,j} (F_{ij}^{\ell} - P_{ij}^{\ell})^2$$

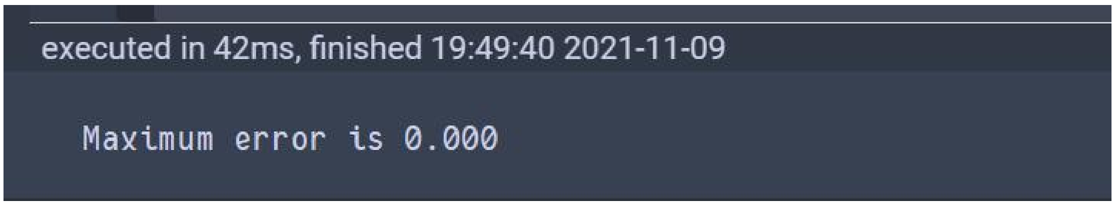
Code:

content_loss 按照提示写代码



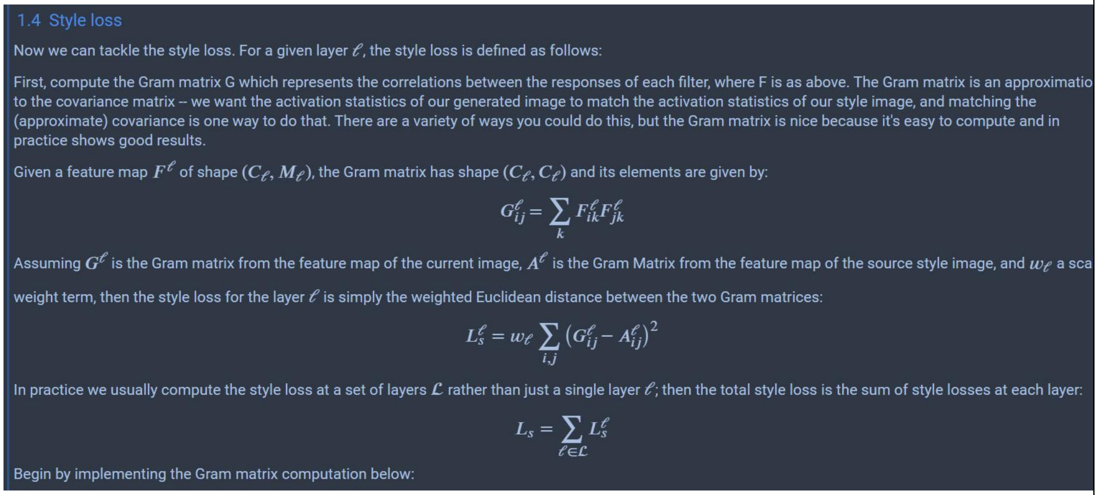
```python
def content_loss(content_weight, content_current, content_original):
    """
    Compute the content loss for style transfer.

    Inputs:
    - content_weight: Scalar giving the weighting for the content loss.
    - content_current: features of the current image; this is a PyTorch Tensor of shape
      (1, C_l, H_l, W_l).
    - content_target: features of the content image, Tensor with shape (1, C_l, H_l, W_l).

    Returns:
    - scalar content loss
    """
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

    loss_content = content_weight * torch.sum((content_current-content_original)**2)
    return loss_content

    # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

Result:正确



```
executed in 42ms, finished 19:49:40 2021-11-09

Maximum error is 0.000
```

2.style loss



1.4 Style loss

Now we can tackle the style loss. For a given layer ℓ, the style loss is defined as follows:

First, compute the Gram matrix G which represents the correlations between the responses of each filter, where F is as above. The Gram matrix is an approximatio to the covariance matrix -- we want the activation statistics of our generated image to match the activation statistics of our style image, and matching the (approximate) covariance is one way to do that. There are a variety of ways you could do this, but the Gram matrix is nice because it's easy to compute and in practice shows good results.

Given a feature map F^ℓ of shape (C_ℓ, M_ℓ), the Gram matrix has shape (C_ℓ, C_ℓ) and its elements are given by:

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

Assuming G^ℓ is the Gram matrix from the feature map of the current image, A^ℓ is the Gram Matrix from the feature map of the source style image, and w_ℓ a sca weight term, then the style loss for the layer ℓ is simply the weighted Euclidean distance between the two Gram matrices:

$$L_s^\ell = w_\ell \sum_{i,j} (G_{ij}^\ell - A_{ij}^\ell)^2$$

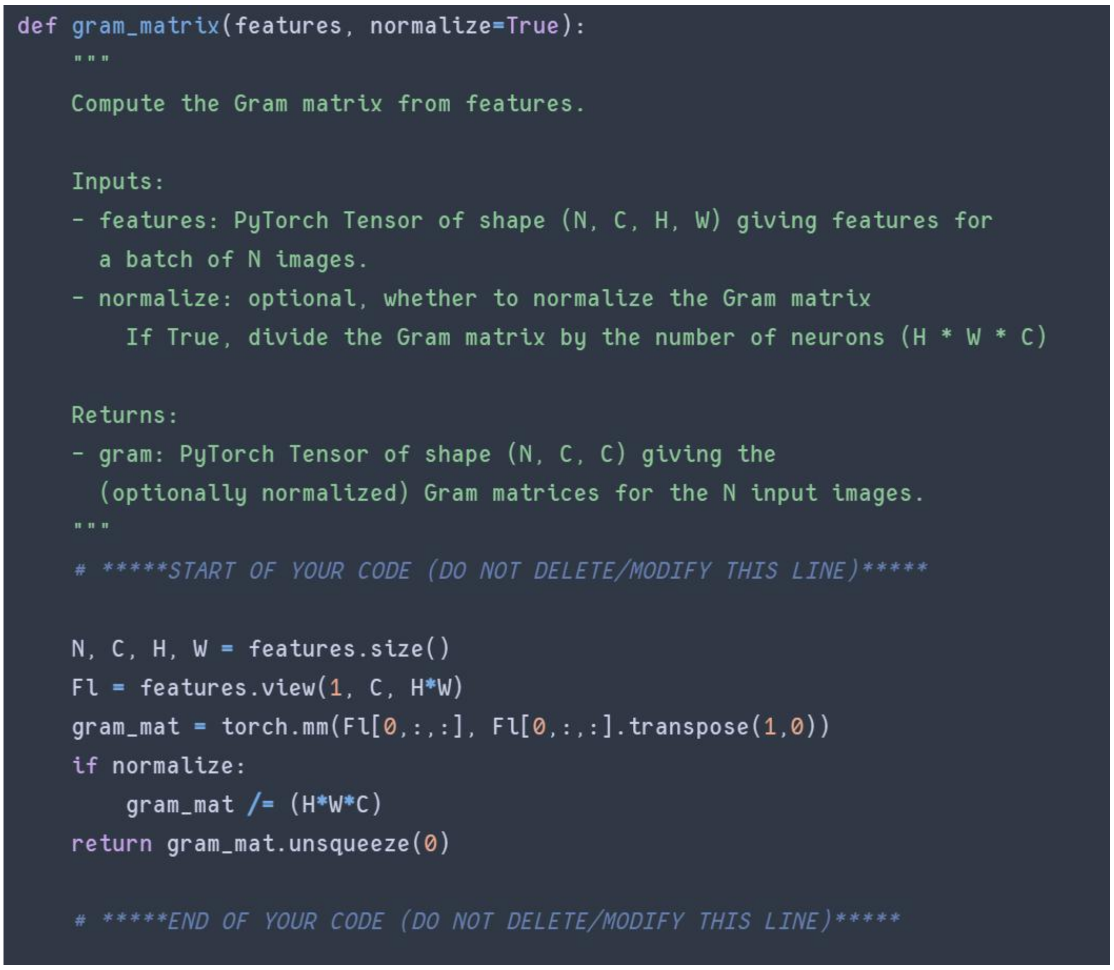
In practice we usually compute the style loss at a set of layers $\mathcal{L}$ rather than just a single layer ℓ; then the total style loss is the sum of style losses at each layer:

$$L_s = \sum_{\ell \in \mathcal{L}} L_s^\ell$$

Begin by implementing the Gram matrix computation below:

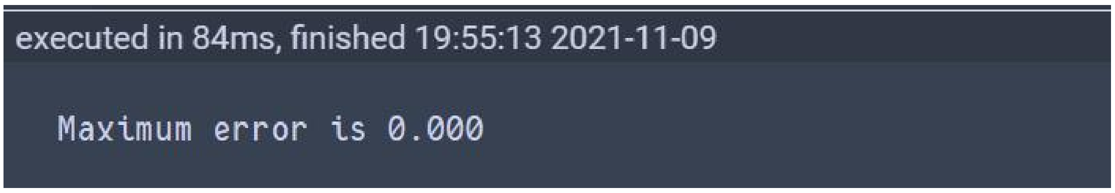
Code:

gram_matrix:按照提示写代码，使用 torch.mm 实现矩阵相乘，然后注意维度对齐和下标对应，以及处理归一化



```python
def gram_matrix(features, normalize=True):
    """
    Compute the Gram matrix from features.

    Inputs:
    - features: PyTorch Tensor of shape (N, C, H, W) giving features for
      a batch of N images.
    - normalize: optional, whether to normalize the Gram matrix
        If True, divide the Gram matrix by the number of neurons (H * W * C)

    Returns:
    - gram: PyTorch Tensor of shape (N, C, C) giving the
      (optionally normalized) Gram matrices for the N input images.
    """
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

    N, C, H, W = features.size()
    Fl = features.view(1, C, H*W)
    gram_mat = torch.mm(Fl[0,:,:], Fl[0,:,:].transpose(1,0))
    if normalize:
        gram_mat /= (H*W*C)
    return gram_mat.unsqueeze(0)

    # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

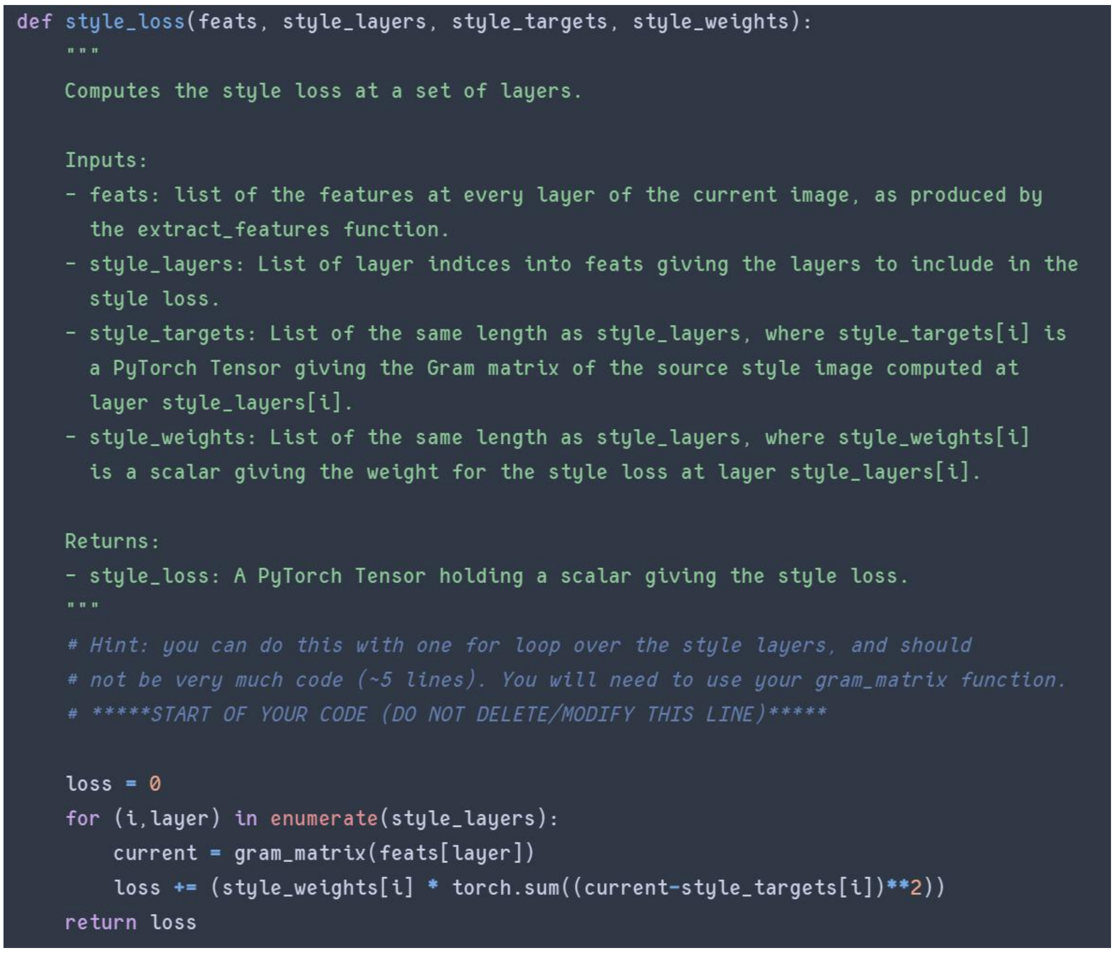
Result:正确



```
executed in 84ms, finished 19:55:13 2021-11-09

Maximum error is 0.000
```

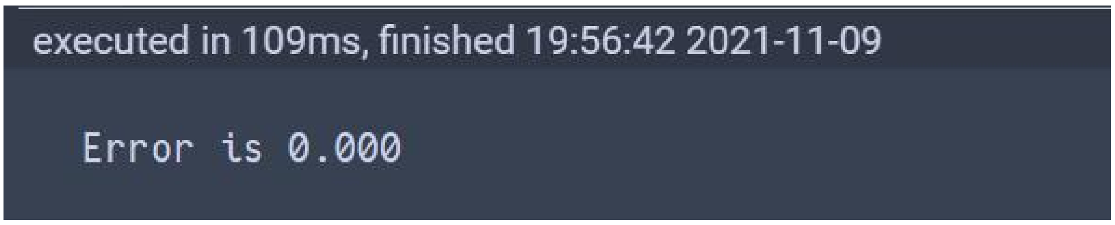
Code:

Style_loss:按照提示写代码，注意使用 gram_matrix



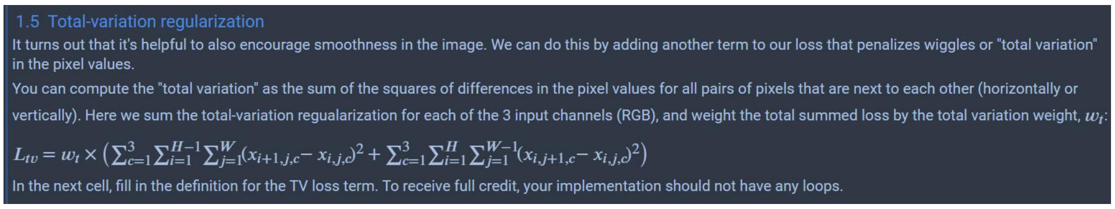
```python
def style_loss(feats, style_layers, style_targets, style_weights):
    """
    Computes the style loss at a set of layers.

    Inputs:
    - feats: list of the features at every layer of the current image, as produced by
      the extract_features function.
    - style_layers: List of layer indices into feats giving the layers to include in the
      style loss.
    - style_targets: List of the same length as style_layers, where style_targets[i] is
      a PyTorch Tensor giving the Gram matrix of the source style image computed at
      layer style_layers[i].
    - style_weights: List of the same length as style_layers, where style_weights[i]
      is a scalar giving the weight for the style loss at layer style_layers[i].

    Returns:
    - style_loss: A PyTorch Tensor holding a scalar giving the style loss.
    """
    # Hint: you can do this with one for loop over the style layers, and should
    # not be very much code (~5 lines). You will need to use your gram_matrix function.
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

    loss = 0
    for (i,layer) in enumerate(style_layers):
        current = gram_matrix(feats[layer])
        loss += (style_weights[i] * torch.sum((current-style_targets[i])**2))
    return loss
```

Result:正确



```
executed in 109ms, finished 19:56:42 2021-11-09

Error is 0.000
```

3.total-variation regularization



1.5 Total-variation regularization

It turns out that it's helpful to also encourage smoothness in the image. We can do this by adding another term to our loss that penalizes wiggles or "total variation" in the pixel values.

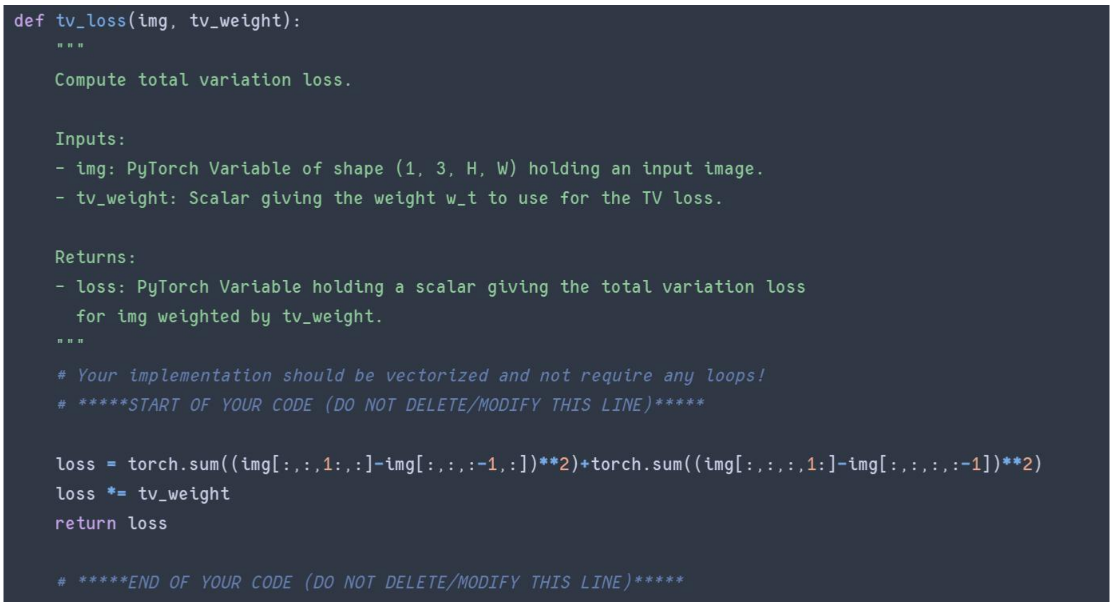
You can compute the "total variation" as the sum of the squares of differences in the pixel values for all pairs of pixels that are next to each other (horizontally or vertically). Here we sum the total-variation regualarization for each of the 3 input channels (RGB), and weight the total summed loss by the total variation weight, w_t:

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

In the next cell, fill in the definition for the TV loss term. To receive full credit, your implementation should not have any loops.

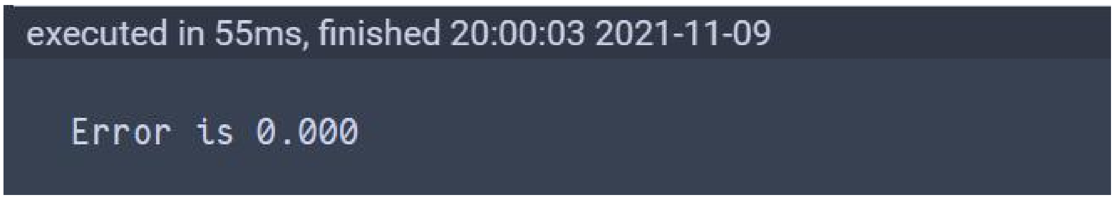
Code:

tv_loss:按照提示写代码，注意可以使用切片，不需要使用循环，就是要注意变量范围即可



```python
def tv_loss(img, tv_weight):
    """
    Compute total variation loss.

    Inputs:
    - img: PyTorch Variable of shape (1, 3, H, W) holding an input image.
    - tv_weight: Scalar giving the weight w_t to use for the TV loss.

    Returns:
    - loss: PyTorch Variable holding a scalar giving the total variation loss
      for img weighted by tv_weight.
    """
    # Your implementation should be vectorized and not require any loops!
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

    loss = torch.sum((img[:,:,1:,:]-img[:,:,:-1,:])**2)+torch.sum((img[:,:,:,1:]-img[:,:,:,:-1])**2)
    loss *= tv_weight
    return loss

    # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

Result:正确

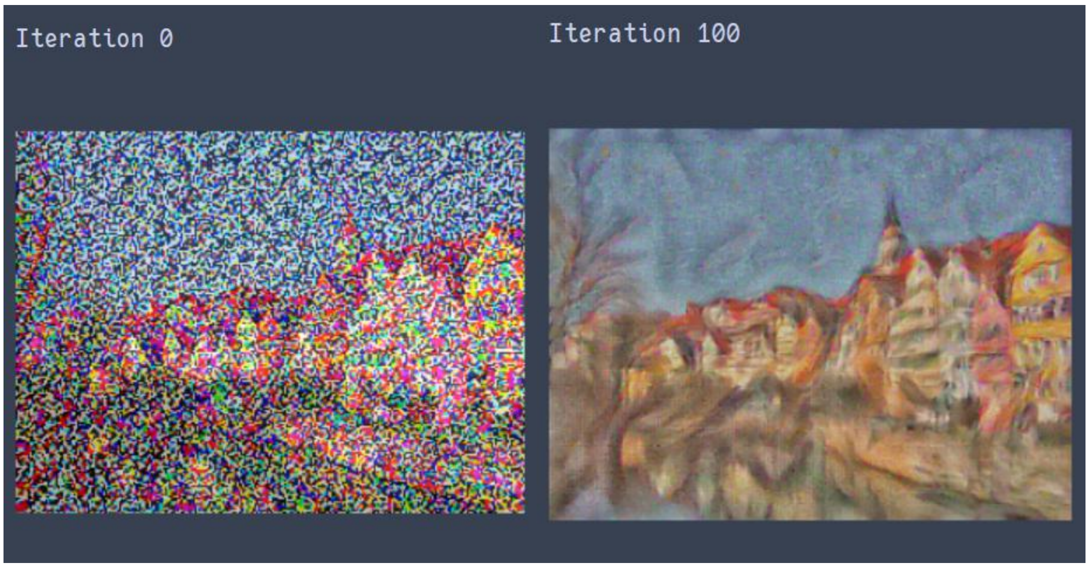


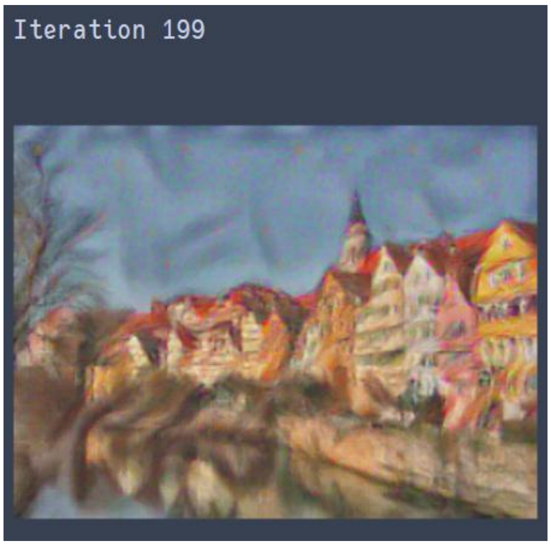
```
executed in 55ms, finished 20:00:03 2021-11-09

Error is 0.000
```

多种不同风格转换的实现可视化:

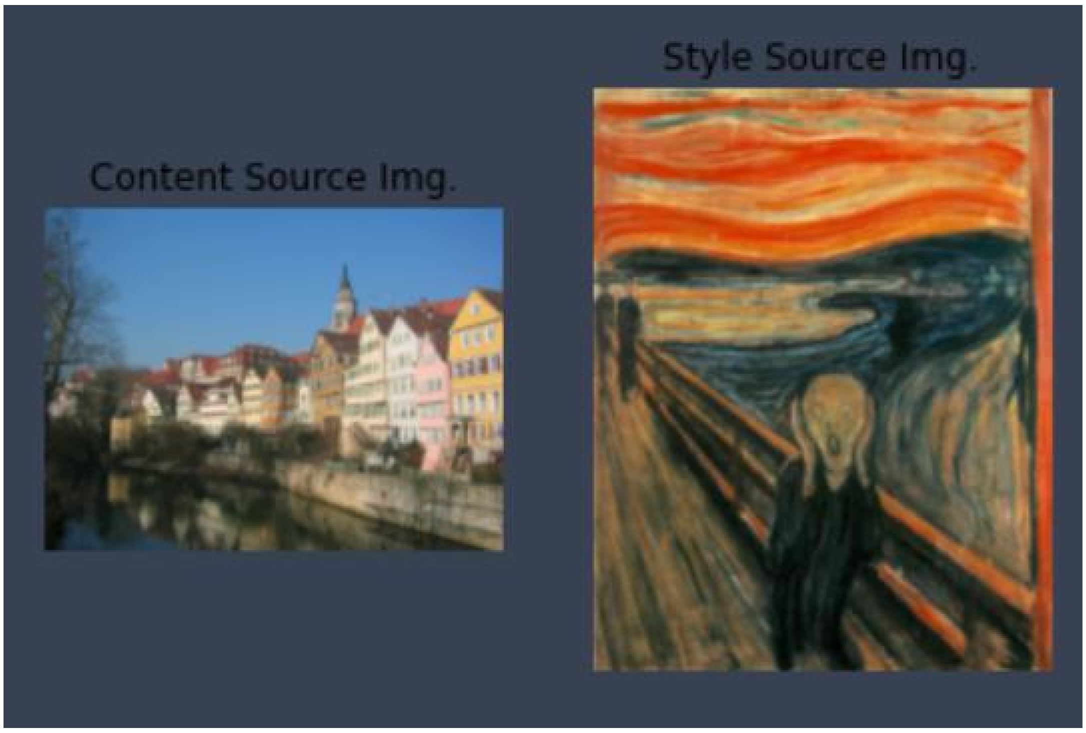
```
# Composition VII + Tubingen
```

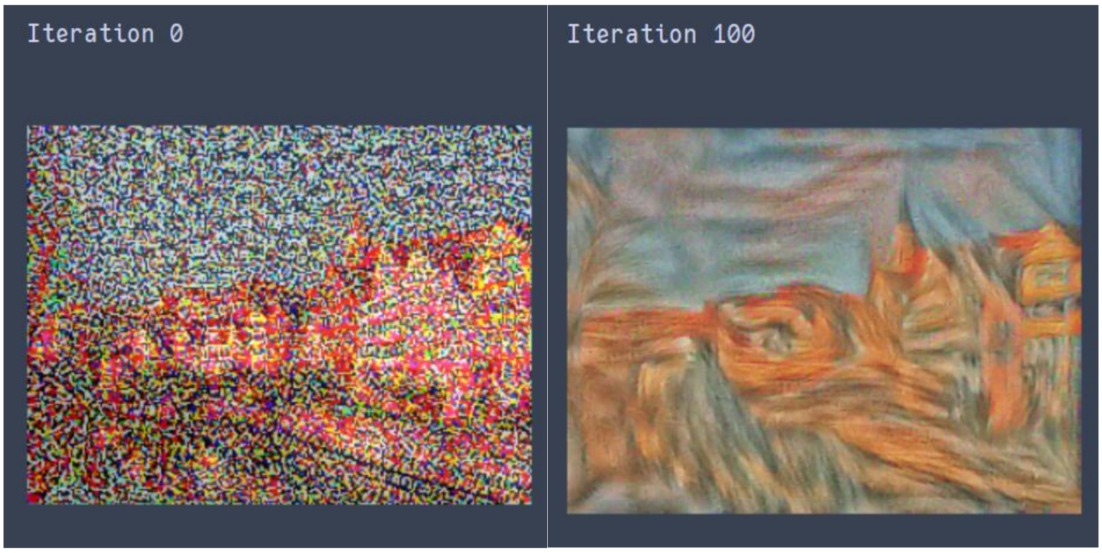
Content Source Img.
Style Source Img.





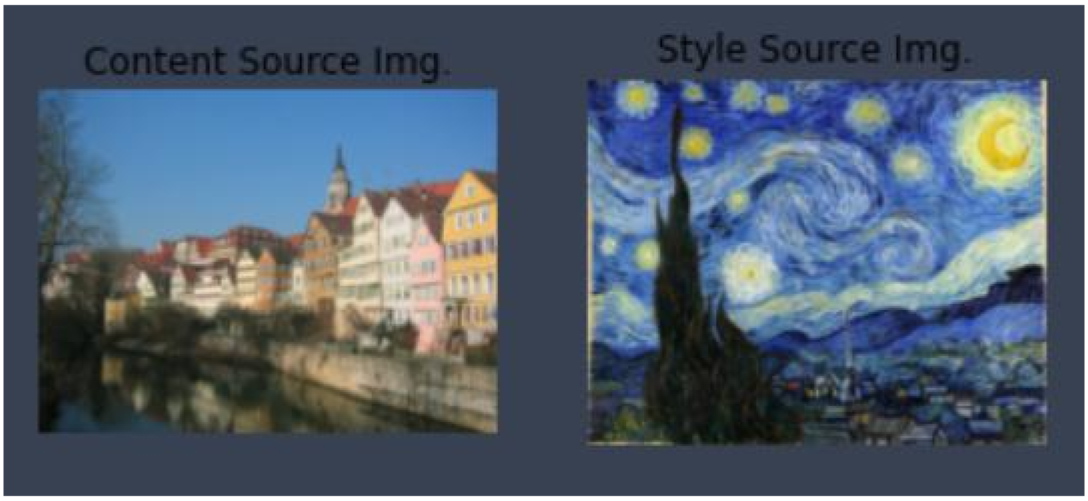
```
# Scream + Tubingen
```

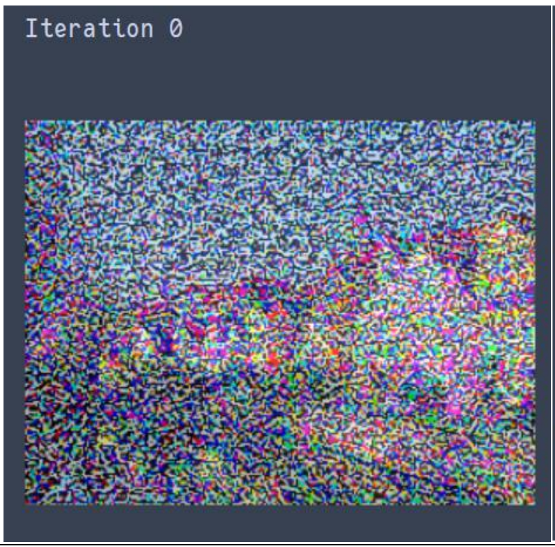


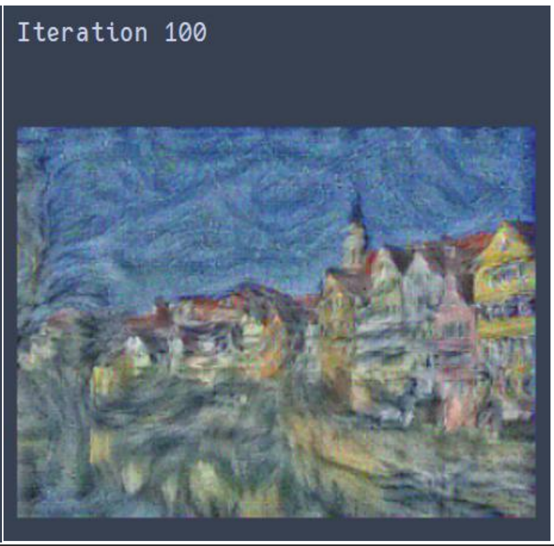


Iteration 199

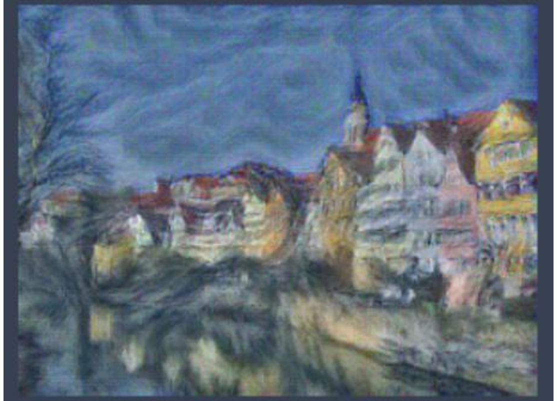
# Starry Night + Tubingen



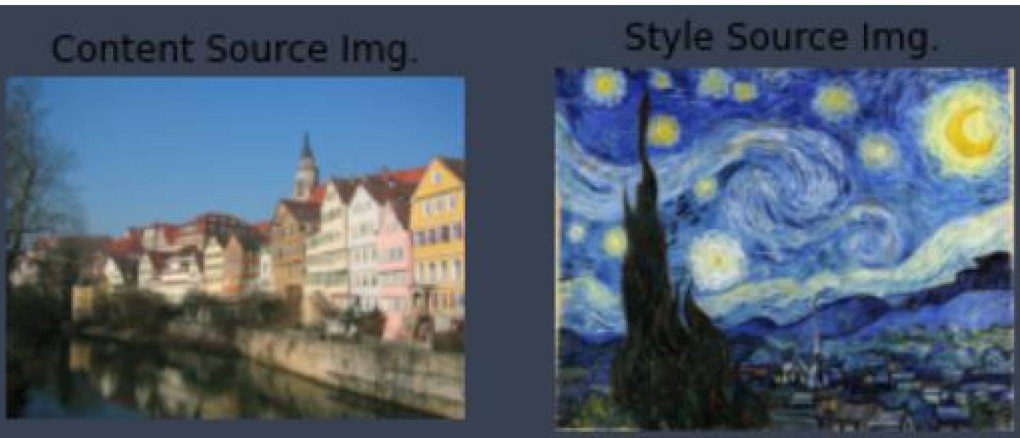




Iteration 199

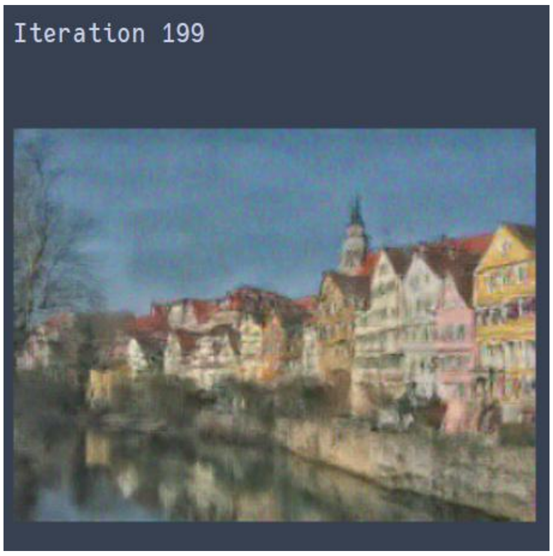


```
# Feature Inversion -- Starry Night + Tubingen
```



Iteration 0

Iteration 100

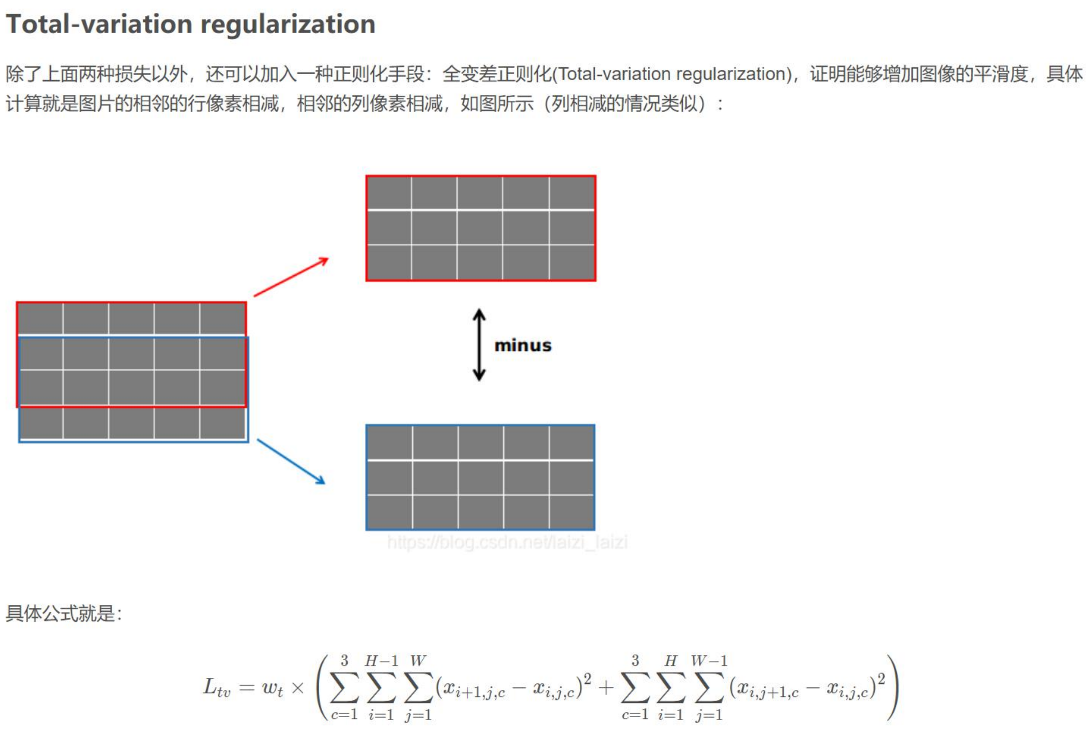


结论分析与体会：

・理解风格迁移：

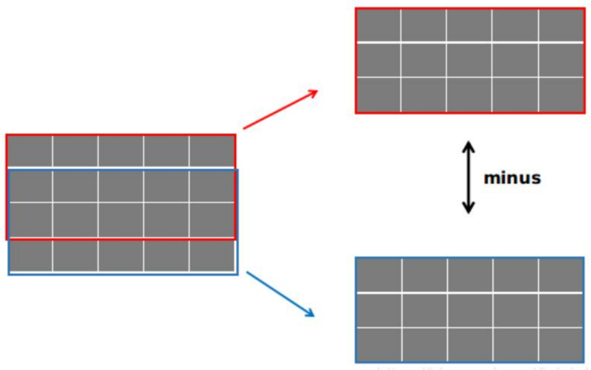
拿两张图像，一张为内容图像(content source)，一张为风格图像(style source)，然后生成一张图像。通过构造一个总的损失函数，减小损失使生成的图像能够匹配内容特征与网络各层的风格特征，对生成图像像素进行梯度下降，减小总损失，就能使生成的图像既具有内容图像的内容，也有风格图像的风格。

・形象理解 tv_loss



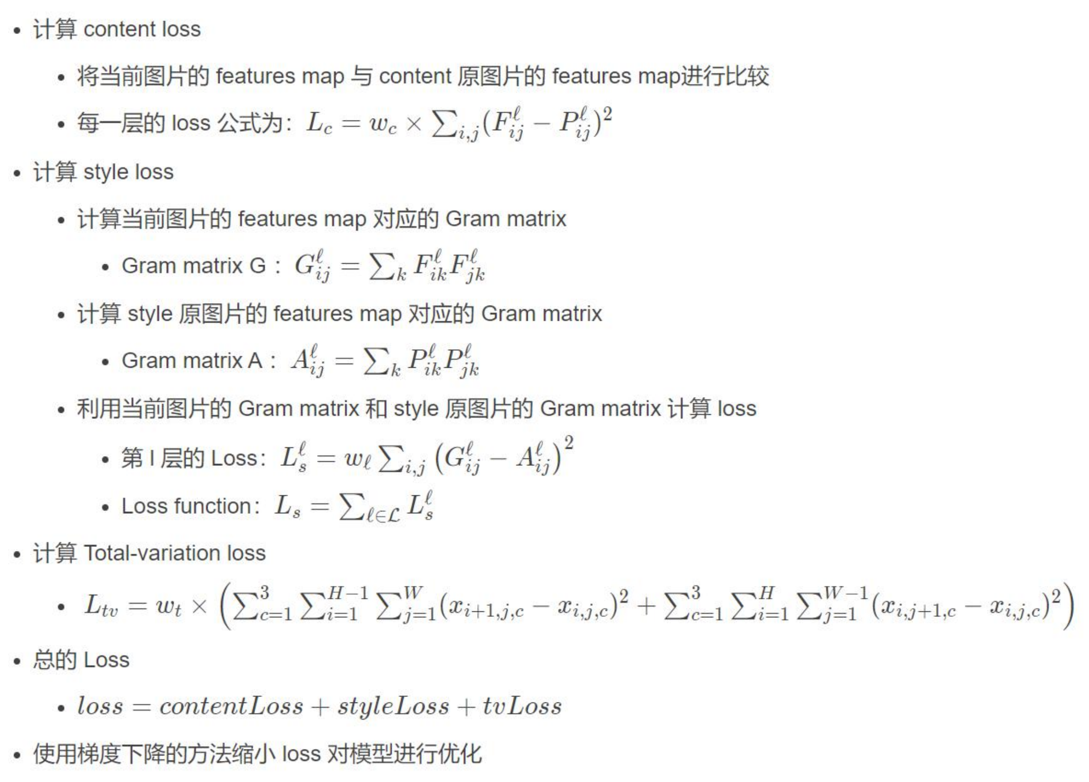
Total-variation regularization

除了上面两种损失以外，还可以加入一种正则化手段：全变差正则化(Total-variation regularization)，证明能够增加图像的平滑度，具体计算就是图片的相邻的行像素相减，相邻的列像素相减，如图所示（列相减的情况类似）：



具体公式就是：

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



- 计算 content loss
  - 将当前图片的 features map 与 content 原图片的 features map进行比较
  - 每一层的 loss 公式为：$L_c = w_c \times \sum_{i,j}(F_{ij}^{\ell} - P_{ij}^{\ell})^2$
- 计算 style loss
  - 计算当前图片的 features map 对应的 Gram matrix
    - Gram matrix G：$G_{ij}^{\ell} = \sum_k F_{ik}^{\ell} F_{jk}^{\ell}$
  - 计算 style 原图片的 features map 对应的 Gram matrix
    - Gram matrix A：$A_{ij}^{\ell} = \sum_k P_{ik}^{\ell} P_{jk}^{\ell}$
  - 利用当前图片的 Gram matrix 和 style 原图片的 Gram matrix 计算 loss
    - 第 l 层的 Loss：$L_s^{\ell} = w_{\ell} \sum_{i,j} \left(G_{ij}^{\ell} - A_{ij}^{\ell}\right)^2$
    - Loss function：$L_s = \sum_{\ell \in \mathcal{L}} L_s^{\ell}$
- 计算 Total-variation loss
  - $L_{tv} = w_t \times \left(\sum_{c=1}^{3} \sum_{i=1}^{H-1} \sum_{j=1}^{W} (x_{i+1,j,c} - x_{i,j,c})^2 + \sum_{c=1}^{3} \sum_{i=1}^{H} \sum_{j=1}^{W-1} (x_{i,j+1,c} - x_{i,j,c})^2\right)$
- 总的 Loss
  - $loss = contentLoss + styleLoss + tvLoss$
- 使用梯度下降的方法缩小 loss 对模型进行优化

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：

1，一开始没有理解三种 loss 分别的作用，后来深入学习之后发现：

Content loss:生成图像与内容源图像某一层 feature map 之间的内容 deviation

Style loss：生成图像与风格源图像某几层 feature map 之间的风格 deviation

Total-variation regularization：全变差正则化，证明能够增加图像的平滑度

2，完整的风格迁移过程：

```
初始化 img 图片（可以初始化为随机噪声或从另一张图片复制）
for t in （迭代次数）
  使用 cnn 获取 feature map
  计算 img 的总 loss（content loss+style loss+tv loss）
  反向传播计算 img 的梯度
  更新 img
```