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： 生成新图像(网络可视化)和风格转换		学号：202020130190
日期：2022.10.17	班级： 人工智能	姓名：刘绪波
Email：2842353032@qq.com		

实验目的：

在本作业中，将探索在 ImageNet 上可视化预训练模型特征的方法，以及实现样式传递的此模型。 探索图像渐变的各种应用，包括 saliency maps, fooling images, class visualizations；了解并实施图像风格转换技术

实验软件和硬件环境：

软件: Dataspell 2022.2.2 ；

硬件：CPU：AMD Ryzen 7 4800U; 显卡：AMD Radeon

实验原理和方法：

1. 生成新图像(网络可视化)：

① Saliency maps:

用来做模型的解释，可以用来知道哪些变量对于模型来说是重要的。我们也可以理解为 Saliency map 即特征图，可以告诉我们图像中的像素点对图像分类结果的影响。

数学概念来解释：寻找哪些像素点对图像分类得分影响最大->梯度。得分对像素的梯度大时，说明该像素对得分的影响大。

② Fooling images:

该方法其实就是对一张图片进行处理，也就是添加一点随机噪声，让肉眼看起来没什么特别的变化，仍然是原来的类别，但是因为这么一点噪声，让神经网络识别成了其他的类别的东西；也就是说，利用图像梯度来生成欺骗神经网络的图片。给定输入图片 X（比如猫），目标类别 Y（比如狗），利用梯度上升来更新图片 X 从而最大化得分函数，使得网络将图片 X’分类为 Y。

③ Class visualization:

在第二类问题的基础上，Class visualization 对目标函数增加了一些正则项，提高了产生的图片的质量。公式如下：

$$I^* = \arg\max_I (s_y(I) - R(I))$$

$$R(I) = \lambda \|I\|_2^2$$

2. 图像风格转换：

基本的思路大概是输入两张图 A 和 B。风格迁移是生成图片 C，既可以保留图片 A 的内容，同时将图片 A 的风格转换/迁移为图片 B 的风格。因此，设计一个包含多个目标的目标函数，其中第一项计算图片 C 与图片 A 的内容的偏差，第二项计算图片 C 与图片 B 的风格差异，第三项是图片 C 的正则项，使输出图片平滑；关键在于几个 loss 的设计；下面具体步骤中有所介绍；

另外，我们用作特征提取器的深度网络是 SqueezeNet，之所以选择这个，就是因为它体积小，效

率高。

Loss 计算：content loss，style loss，total-variation loss;

① Content loss：

Then the content loss is given by:

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

② Style loss：

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

③ Total-variation loss:

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

实验步骤：（不要求罗列完整源代码）

首先，启动 IPython，打开文件夹 homework_4，找到　文件：阅读补全代码：

1. 生成新图像(网络可视化)：

① 设计了一写对于数据预处理有帮助的函数；

② 数据预处理模型：对于图像生成实验，我们将从一个卷积神经网络开始，该网络经过预训练，在 ImageNet 上进行图像分类。在本次实验上采用 SqueezeNet 模型；

③ 加载一些带有正确标签的图像：结果如下：

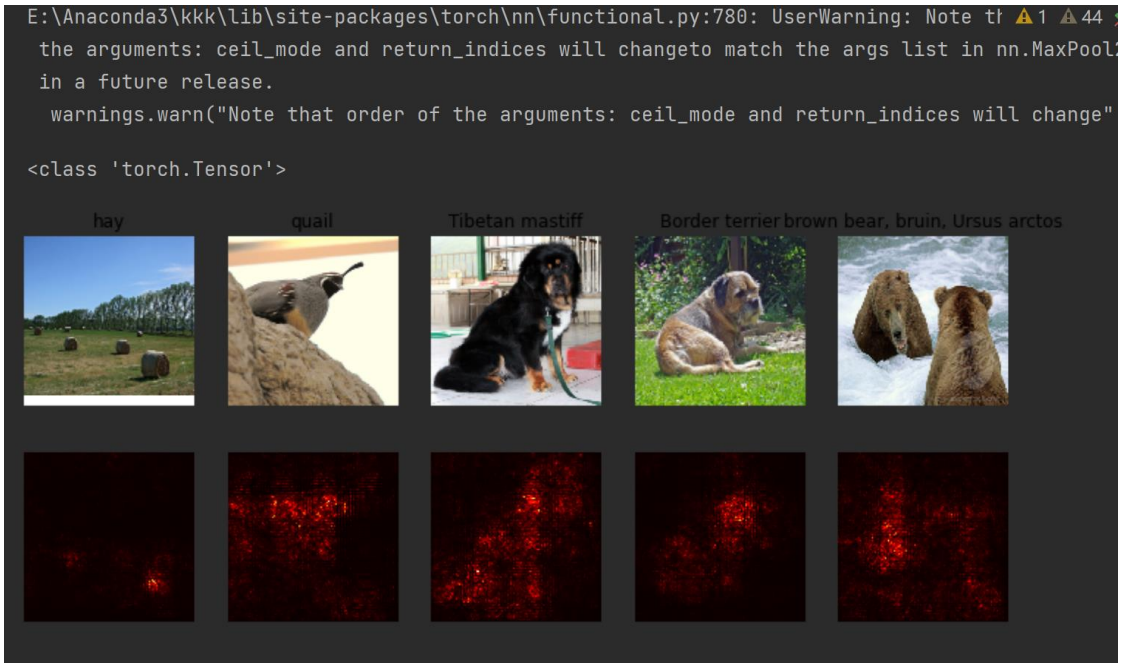
④ 计算 Saliency maps：

计算对应于正确类别的未归一化分数的梯度（它是一个标量），相对于图像的像素。使用 pytorch 的 gather 函数；可运行实例进行观察用法；代码实现：

```
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
s = model(X) # torch.Size([5, 1000])因为是ImageNet，每一个最后都会出来1000类
correct_class_scores = s.gather(1, y.view(-1, 1)).squeeze() # torch.Size([5])
# backward里面也是可以传参数的,而且也不非要是loss才能backward
correct_class_scores.backward(torch.ones_like(correct_class_scores))
saliency, indice2 = torch.max(torch.abs(X.grad), dim=1)
pass

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

⑤ 结果展示：



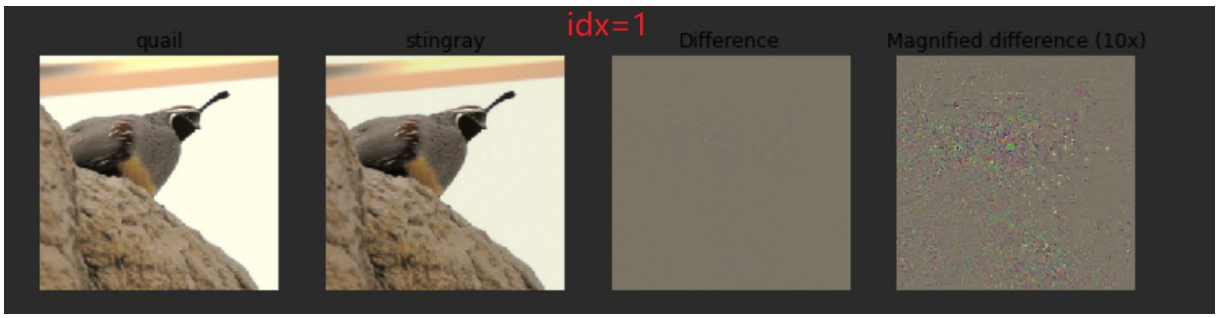
⑥ Fooling images：图像梯度来生成 "欺骗性图像"

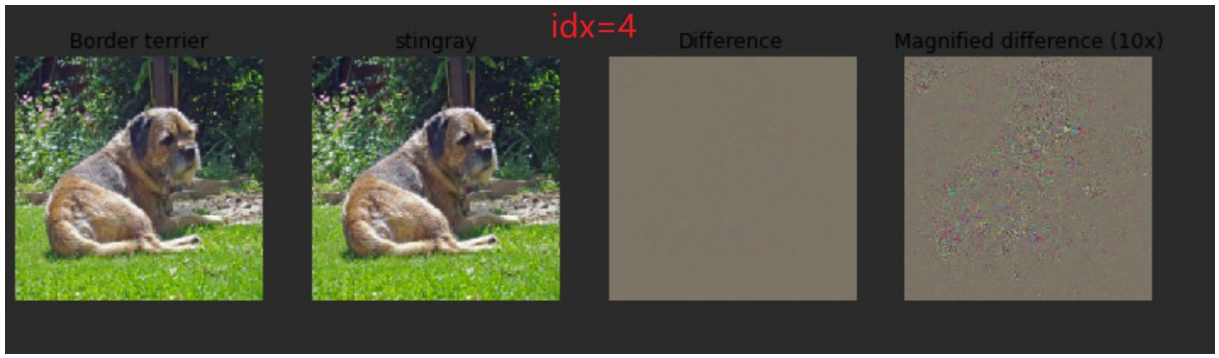
给定一个图像和一个目标类别，我们可以在图像上进行梯度上升以最大化目标类别，当网络将图像分类为目标类别时停止。实现函数来生成欺骗性图像：

```
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
for itera in range(100):
    scores = model(X_fooling) # torch.Size([1, 1000]) 注意不是torch.Size([1000])
    # max返回最大值和它的索引
    if scores.data.max(1)[1].item() == target_y:
        break
    else:
        target_scores = scores[:, target_y]
        # 一般的步骤就是:grad.zero_()-->backward()-->step()
        # 你想优化A，就求A.backward()
        target_scores.backward()
        g = X_fooling.grad.data
        dX = learning_rate * g / torch.norm(g)
        X_fooling.data += dX
        X_fooling.grad.zero_()
pass

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

⑦ 结果展示：所生成的欺骗性图像和原来的十分接近，下面进行可视化区别，并将区别放大十倍观察加入的随机噪声；（通过改变 idx 的值来进行对不同图片的结果进行观察）



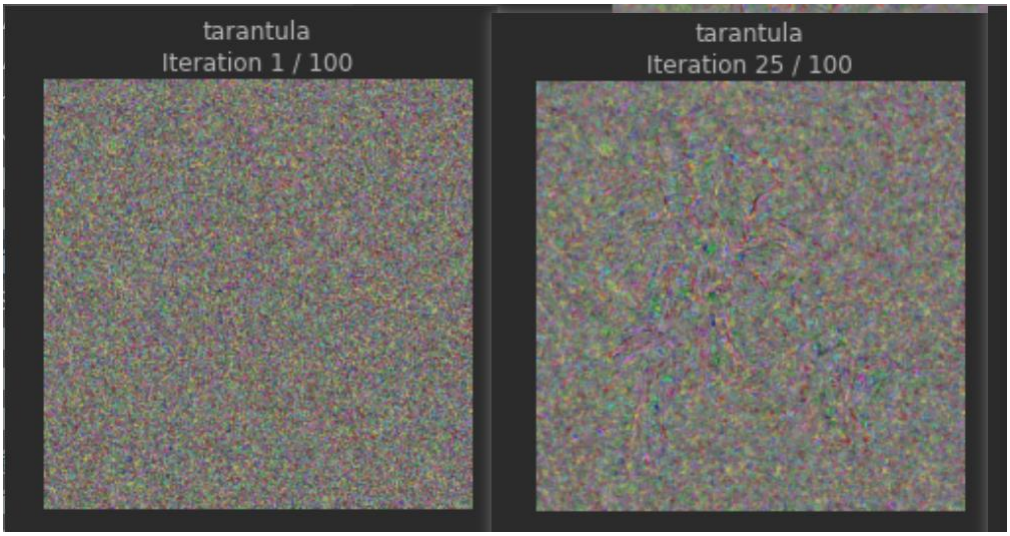


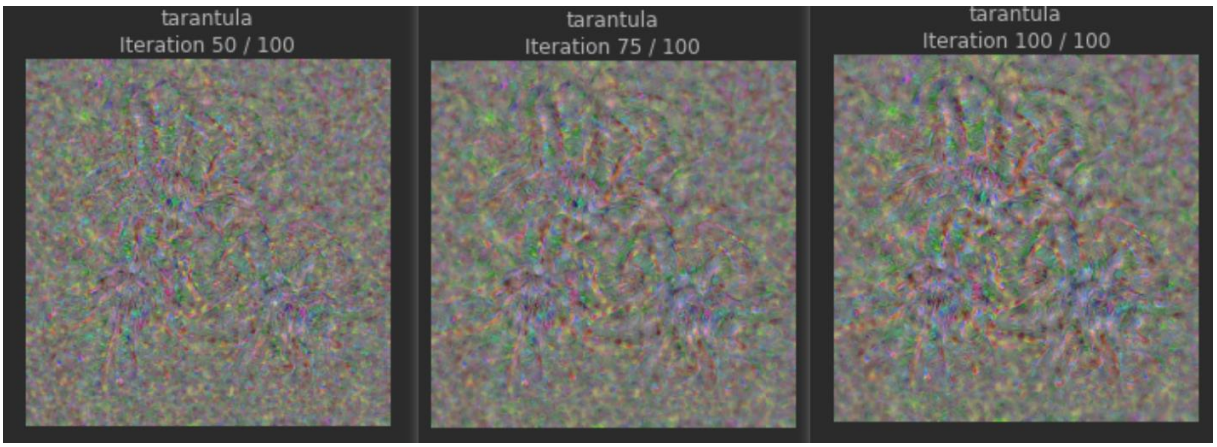
⑧ Class visualization：
生成一个网络可以识别类别的图像（可以用正则化生成一个质量更高的图像）：
代码补全：

```
# Be very careful about the signs of elements in your code.            #
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
scores = model(img)
target_scores = scores[:, target_y]
target_scores.backward()
dX = img.grad.data + 2 *l2_reg * img.data
img.data += learning_rate * dX / torch.norm(dX) # 比之前就多了个正则项
img.grad.zero_()
pass

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

结果展示：





2. 图像风格转换：
   主要介绍几个 loss 的设计：我们现在要计算损失函数的三个组成部分。损失函数是三个项的加权和：内容损失+风格损失+总变化损失；
   ① Content loss:
   代码补全：

```
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    N_l, C_l, H_l, W_l = content_current.shape
    F = content_current.view(C_l, H_l*W_l)
    P = content_original.view(C_l, H_l*W_l)
    ct_loss = content_weight * torch.sum((F - P)**2)
    return ct_loss
    pass

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

然后进行测试所表示代码结果如下：

```
    student_output = content_loss(content_weight, c_feats[content_layer], feats[content_layer]).cpu
     ().data.numpy()
    error = rel_error(correct, student_output)
    print('Maximum error is {:.3f}'.format(error))

content_loss_test(answers['cl_out'])
```

```
Maximum error is 0.000
```

② Style loss:
代码实现：

```
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    sl_loss = 0.0
    for i in range(style_layers.__len__()):
        sl_loss += style_weights[i] * torch.sum((gram_matrix(feats[style_layers[i]].clone()) -
         style_targets[i])**2)
    return sl_loss
    pass

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

结果展示：

```
style_loss_test(answers['sl_out'])
```

```
Error is 0.000
```

③ Total-variation regularization:
代码展示：

```
    # Your implementation should be vectorized and not require any loops!
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    tv1 = torch.sum((img[:,:,:,1:] - img[:,:,:,:-1])**2)
    tv2 = torch.sum((img[:,:,1:] - img[:,:,:-1])**2)
    t_v_loss = tv_weight * (tv1 + tv2)
    return t_v_loss
    pass

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

结果展示：

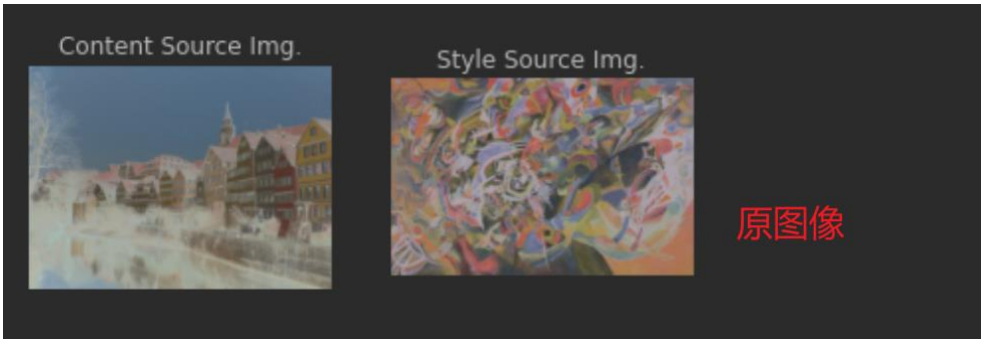
```
tv_loss_test(answers['tv_out'])
```

```
Error is 0.000
```

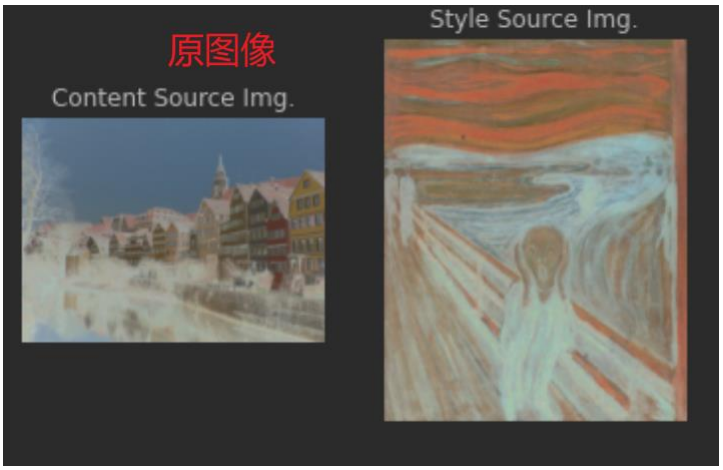
将他们串联起来；

④ 生成一些质量高的图片：

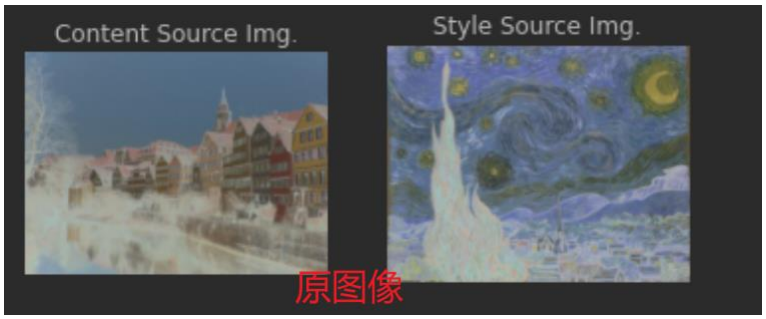
第一组：

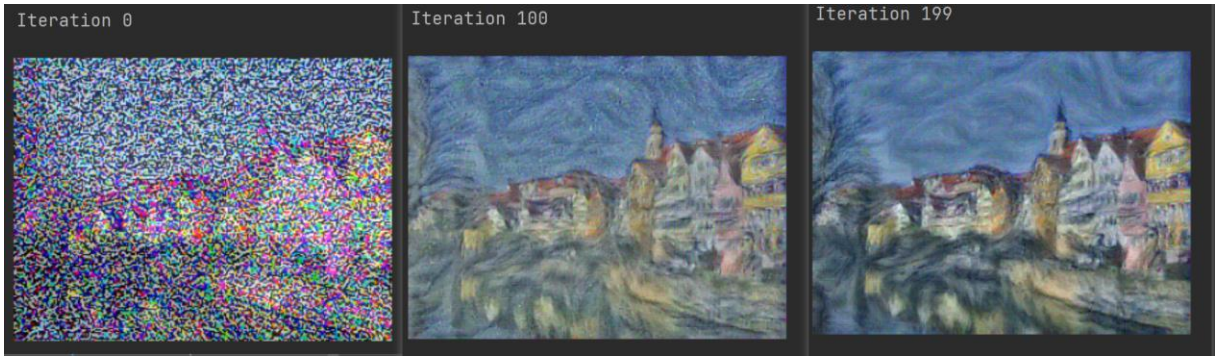


第二组：



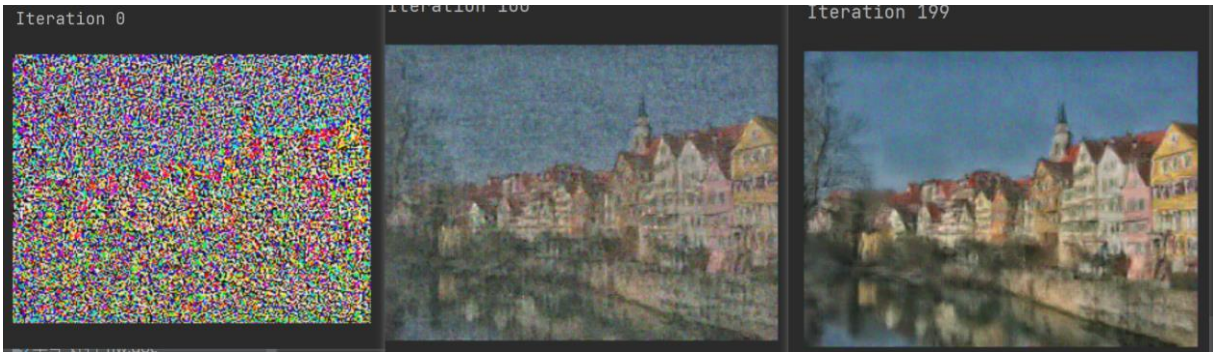
第三组：





⑤ 特征反转：

结果展示：



结论分析与体会：

1. 生成新图像(网络可视化)：
   ① Saliency maps:
   它的一个重要应用就是实现图像中物体的识别和分割：主要步骤：根据原图像来计算得到的 Saliency Maps；然后设定一个阈值，超过某个阈值认为是重要的的元素(可以用不同的元素进行标记出来)；mask 与原始图像结合，得到物体的分割；
   也可以进行目标检测；
   ② Fooling images：
   我们可以对某个图像进行“定向”的“fooling”，也就是说，我们可以不断的用梯度上升来更新图像的像素，直到网络将图像分类为我们指定的“类别”，梯度更新的时候清零，原因就是 tensor.grad()会累加；
   ③ Class visualization：
   正则化方程：

$$I^* = argmax(s_y) - R(I)$$
$$R(I) = \lambda ||I||^2$$

   注意正则项的在优化函数中的符号，是负的，因为我们要最小化正则项 loss。这里可以选用 L2 norm 来做正则化。
2. 图像风格转换：
   ① Content loss ：
   比较 content 的差异，显然不直接对两个图片的 pixel 进行比较，而是比较两个图片的 feature map；首先我们这里再次利用到一个 pre-train 的神经网络。一个已经 pre-train 的神经网络完全可以当做特征提取器。文章利用这个 pre-train 的网络的某些层输出的 feature maps 计算两组 feature map 的差异(L2 norm)作为 content loss。
   ② Style loss：
   将风格定义为 feature map （resize 后为 2D）的协方差矩阵。进而可以用 gram matrix 来近似替代协方差矩阵的计算。
   ③ Total-variation loss:
   实际上就是之前 CS131 图像处理中，分别计算图像沿 x 方向的导数，沿 y 方向的导数的矢量运算，用 neighbor

pixel 的差异做图像平滑。

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. 关于 backward 函数：在实现 compute_saliency_maps 函数时对于 backward 函数的使用错误：
   如果这么写 *correct_class_scores.backward()*会报错
   RuntimeError: grad can be implicitly created only for scalar outputs 即提示我们输出不是一个标量
   原因时里面需要加参数：从而改进成：
   correct_class_scores.backward(torch.ones_like(correct_class_scores))
   然而在 make_fooling_image 中是这样的：

```
target_scores.backward()
g = X_fooling.grad.data
dX = learning_rate * g / torch.norm(g)
X_fooling.data += dX
X_fooling.grad.zero_()
```

   并没有加任何参数；
   因此 pytorch 在求导的过程中，分为下面两种情况：
   如果是标量对向量求导(scalar 对 tensor 求导),那么就可以保证上面的计算图的根节点只有一个，此时不用引入 grad_tensors 参数，直接调用 backward 函数即可
   如果是(向量)矩阵对(向量)矩阵求导(tensor 对 tensor 求导)，实际上是先求出 Jacobian 矩阵中每一个元素的梯度值(每一个元素的梯度值的求解过程对应上面的计算图的求解方法)，然后将这个 Jacobian 矩阵与 grad_tensors 参数对应的矩阵进行对应的点乘，得到最终的结果。