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:Regularization and Batch Normalization		学号：202000130047
日期：2022-10-20	班级： 智能 20	姓名： 夏再禹
Email：842649082@qq.com		

实验目的：

1. 对网络可视化的方法进行实验，补充编写完实验所需代码，研究图像梯度的多种应用，包括 saliency maps, fooling images, class visualizations。

2. 理解并实现图像风格迁移的技术。

实验软件和硬件环境：

软件：jupyter notebook

硬件：cpu: Intel i5

实验原理和方法：

一、Network Visualization

在网络可视化的方法中，一个重要的思想是保持训练好的网络不变，而把输入图像的像素值作为变化的参数，定义相关损失函数并进行反向传播，从而得到我们所期望的图像，或是直接利用梯度进行可视化。

本次网络可视化的三个技术都采用了这种方法，即利用 image gradients 来生成新的图像。

1. Saliency Maps

Saliency Maps 是一个判断图像上哪部分影响了网络的分类结果，以及每一部分的影响程度。

具体来说，它会告诉我们每个像素对分类结果的影响程度。它是如何衡量每个像素的影响程度呢？答案是采用 image gradients，对模型输出的相应类别的概率或得分进行求导，我们可以得到图像上每个像素值的梯度，梯度越大，则代表其变化一定量后对结果影响越大，也就是其对分类结果的影响程度更大。

因此计算过程为：先对模型输出的相应类别的概率或得分进行反向传播，我们会得到一个大小为$3\times H\times W$的 image gradients(与原图同大小)，然后我们对其求绝对值，再对每个像素的三个通道的梯度绝对值，取其中最大的值，从而即可得到一个大小为$H\times W$的图像，即为一个 saliency map。

2. Fooling Images

我们可以通过对一个图像的像素值进行轻微的调整，使其被网络重认为是其它类别,而我们人眼看会觉得它没什么变化,这个调整后的会被网络认错的图像即为 Fooling image。

要生成 Fooling image 图像很简单，以一个错误类别的模型给出的得分作为优化目

标，以其进行反向传播，输入的图像作为优化变量，用 image gradient 对其进行优化即可。

3. Class visualization

我们可以生成一个图像来最大化网络输出的某个类别的得分，这可以先随机初始化一个图像，然后通过利用 image gradient 进行梯度上升来生成。

通过 Class visualization，我们可以体会网络认为什么样的图像是这个类别的。

二、Style Transfer

Style Transfer 即利用两张图像，生成一个内容同一张图片相同，风格同另一张图片相同的图片，即把一张图片的风格迁移到另一张图片上去。

可以通过定义一个损失函数包括对代表一张图片的内容和代表另一张图片的风格的特征图进行匹配，然后对图片的像素进行梯度下降来实现。

而对于损失函数的定义，即如何表征内容与风格，核心思想为直接计算两张图像的像素差异可以获得内容损失，计算两种图像的特征图协方差矩阵可以获得风格损失。

实验步骤：（不要求罗列完整源代码）

一、Network Visualization

1. Saliency Map

主要需要完成一个计算 saliency map 的函数。

这里函数输入为一个 Minibatch 的数据，因此输入图像大小为 $N \times 3 \times H \times W$，（1）首先我们将其输入到模型中

（2）然后在输出中提取相应类别的得分之和，这里要使用 gather 函数。

（3）对总得分进行反向传播

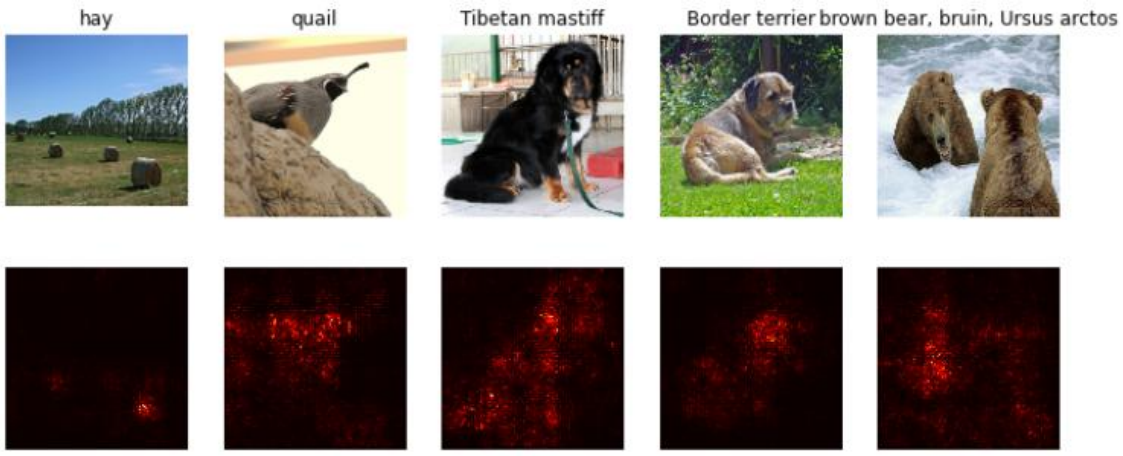
（4）最后将 image gradient 求绝对值再求三通道上的最大值。

补充代码为：

```
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
predict_y = model(X)
loss = torch.sum(predict_y.gather(1,y.view(-1,1)).squeeze())
loss.backward()
saliency,_ = torch.abs(X.grad).max(dim=1)
pass

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

输出的 Saliency Maps 如下：



从 Saliency Maps 中可见，类别对应的物体，如狗、熊，在 Saliency Maps 中最显著，而背景的影响不大，则说明此网络模型确实是对这些物体进行了判断。

Inline question:

A friend of yours suggests that in order to find an image that maximizes the correct score, we can perform gradient ascent on the input image, but instead of the gradient we can actually use the saliency map in each step to update the image. Is this assertion true? Why or why not?

我认为不行，因为 Saliency Maps 对梯度求了绝对值，又对三通道取了最大值，因此我们不知道对图像值加还是减，如果一直加上 Saliency Maps 的值，肯定会很糟糕；同时我们不知道三通道分别应该变化多少，如果我们一直让三通道变化相同的值，抹去了他们的区别，那也会导致结果很糟糕。

2. Fooling Images

主要需要完成一个计算 fooling_image 的函数。

我们需要生成一个与输入图像 X 接近，但被模型认为是类别 target_y 的图像，因此我们可以把要生成的图像先初始化为 X，然后对以下语句不断循环：

（1） 将其输入到模型中

（2） 判断模型预测的类别（即得分最高的类别）是不是 target_y，若是，退出循环，若不是，进行下一步。

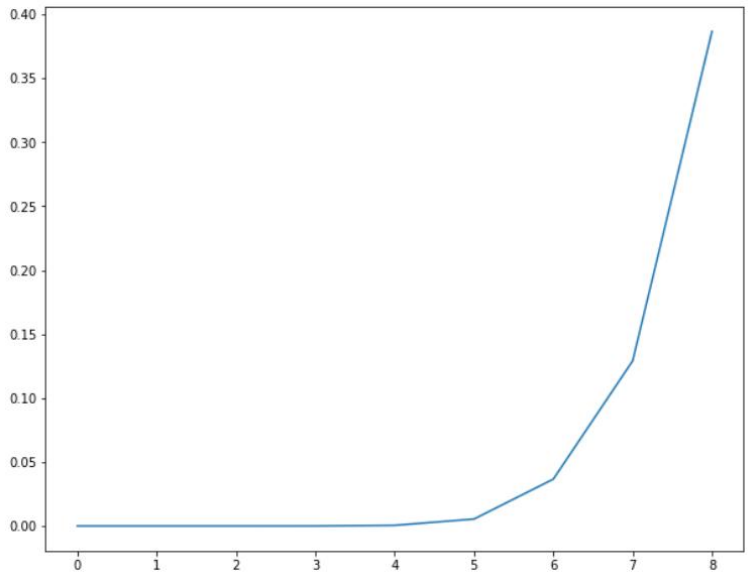
（3）提取模型输出中 target_y 的得分，对其进行反向传播

（4）利用 image gradient 对输入图像进行梯度上升

这里有提示 100 次以内的循环对大多数情况足够，所以写出如下代码：

```
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
model.eval()
p_list = []
for i in range(100):
    predict_y = model(X_fooling).squeeze()
    _,index = predict_y.max(dim=0)
    if index==target_y:
        break
    p = predict_y[target_y]
    p.backward()
    g = X_fooling.grad
    dX = learning_rate * g / g.norm()
    with torch.no_grad():
        X_fooling += dX
        p_list.append(torch.softmax(predict_y,dim=0)[target_y])
    X_fooling.grad.zero_()
plt.plot(p_list)
# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
################################################################################
#                                END OF YOUR CODE
```

生成一个 fooling img，我们可以输出训练过程中相应类别的概率（得分经过 softmax）的变化：



可见只经过了 9 次迭代，fooling img 被判断是类别 target_y 的概率就以及接近 0.4 并成为最高的了。

输出原图、fooling img、这两张图片的差、10 倍放大像素值后的这两张图片的差如下：

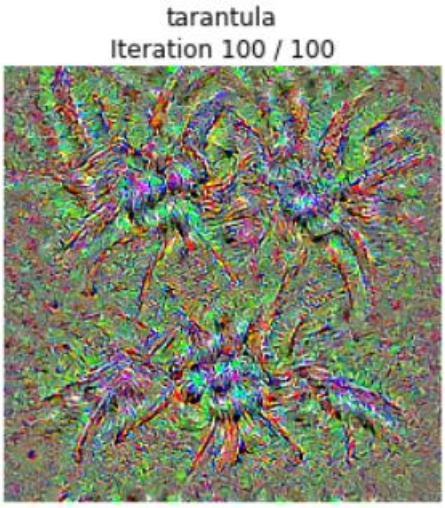
可见肉眼几乎看不出 fooling img 与原图有什么差别，但网络就是弄错了，显示出了 CNN 的一个局限性。放大他们的差异，可见在 hay 出现的地方差异会比较大，这也给网络了一个可视化的解释，说明网络比较关注 hay 出现的地方。

3. Class visualization

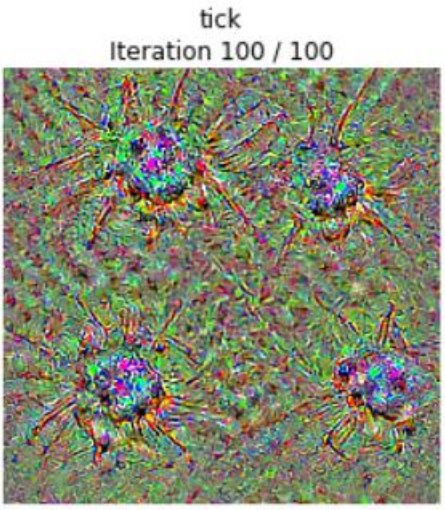
我们主要完成一个生成最大化类别 target_y 的分数的图像的函数，这里还加入了正则项。
我们需完成的步骤很简单，先将图像传入模型，提取出所需类别的得分，并加入计算出的正则项，进行反向传播，然后再利用 img gradient 让图像进行梯度上升即可。
补充代码为：

```
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
predict = model(img).squeeze()
p = predict[target_y] - l2_reg*img.norm()**2
p.backward()
with torch.no_grad():
    img += learning_rate*img.grad
img.grad.zero_()
model.zero_grad()
# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

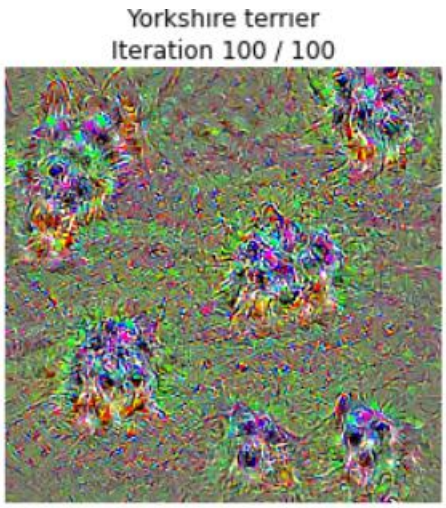
迭代一百次后，结果为：



狼蛛



扁虱



约克夏梗（一种长毛小狗）

观察 class visualization，可见它们都捕捉了对应类别的某种特征，并给出了它们的部分形态，从狼蛛的图中，可看到狼蛛的众多腿四处张开的形态，且腿比中间的身子长很多，而扁虱的图则显示出其腿较细，身子较大。约克夏梗的图显示了其脸部的形态，我们可看出其脸部的轮廓，以及其耳朵、眼睛、鼻子、嘴巴。

同时每个 class visualization 都含有多个对应类别的物体，且因为每块局部的初始化不一样，就可能得到了不一样的角度观察的物体，这在约克夏梗的图中尤为明显，每个脸的角度都不同。

二、 Style Transfer

1. content loss

Content loss 就是计算特征图之间 L2 距离，比较简单。
补全代码为：

```
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
return content_weight*((content_current-content_original).norm()**2)

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

2. style loss

Style loss 需要计算 Gram 矩阵，先完成算 Gram 矩阵的函数：

其实就是矩阵乘以其转置，这里因为输入的 features 矩阵有四维，(N,C,H,W)，我们需要先把最后两个维度拉平为一个维度，然后再使用矩阵乘法，此时转置使用方法 swapdims(1,2)，因为我们需对最后两维转置，而第 0 维不用动。

补全代码为:

```
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
N,C,H,W = features.shape
features = features.reshape(N,C,-1)
gram = torch.bmm(features,features.swapdims(1,2))
if normalize:
    gram /= H*W*C
return gram
# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

然后完成计算 style loss 的函数：

我们需要把多层的损失加起来，每层的损失用生成图的该层特征图的 Gram 矩阵与提供风格图的该层特征图的 Gram 矩阵的 L2 矩阵，并乘以该层的权重。

补全代码如下：

```
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
loss = 0
for i,si in enumerate(style_layers):
    gram_current = gram_matrix(feats[si])
    loss += style_weights[i] * (gram_current-style_targets[i]).norm()**2

return loss

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

3. Total-variation regularization

这里我们加入一个正则项，定义为：

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

通过加入此正则项，可使图像更平滑。我们可以通过切片后的矩阵的 L2 距离来计算此正则项，代码为：

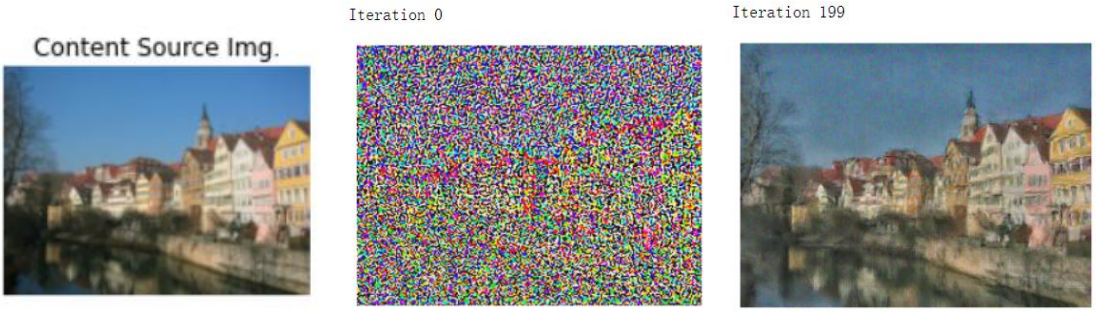
```
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
N,C,H,W = img.shape
L = tv_weight*((img[:,:,:(H-1),:]-img[:,:,1:,:]).norm()**2 + (img[:,:,:,:(W-1)]-img[:,:,:,1:]).norm()**2)
return L
# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

Style transfer 结果：

可见生成图像的内容与 content source 相近，风格与 style source 相近。

Feature Inversion

我们可以利用上面 style transfer 的方法，但把损失函数中，风格损失的权重设置为 0，然后我们用一个随机初始化的图片来进行迭代，最终结果如下：



可见通过特征图，可以把 content source 进行还原，某种程度上反映了特征图的意义。

结论分析与体会：

通过本次实验，体会了三种 Network Visualization 的技术，它们都采用了 image gradient，然后进行了 style transfer 的实现。

首先是 Saliency Map, 其直接对像素的梯度的绝对值进行表示，通过 Saliency Map 可以看出图像上哪块区域对分类结果影响大，可观察网络关注的地方是否正确。

然后是 fooling image, 通过一些对图像的轻微调整，就可以使网络认错其类别，而人眼甚至还看不出调整前后的差异，这反映了神经网络的一个弱点。同时，通过对放大后的调整的观察，我们也可以看出网络更关注哪些地方。

最后是 class visualization，生成了一个使得模型输出的某个类别的得分最高的图片，观察图片就可以知道模型真正捕捉了该类别的什么特征。在实验中，可看出狼蛛的腿、约克夏梗的脸。

对于 style transfer，分别实现了 content loss, style loss 与 Total-variation regularization,通过实验对它们的定义有了更熟悉更清晰的认识，从而也对其原理有了更深刻的认识，其核心就是直接计算两张图像的像素差异可以获得内容损失，计算两种图像的特征图协方差矩阵可以获得风格损失。最后，我们还通过特征图，把 content source 进行了还原，某种程度上反映了特征图的意义。

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.编写 fooling image 的函数的代码时报错“a leaf Variable that requires grad is being used in an in-place operation.”

解决：

若进行一些不需要计算梯度的相关需计算梯度的 tensor 的操作时，需使用 with torch.no_grad()。