Network Visualization (TensorFlow)

在这份Notebook里面我们会来探索图像梯度对于生成新图片的用法。

当训练模型的时候，我们会定义一个损失函数来表示我们对当前模型性能的不满意程度(和标准答案的差异程度)；接着我们用反向传播来计算损失函数对于模型参数的梯度，接着在模型参数上用梯度下降来最小化损失(loss)。

这里我们会做一些不同的事情。我们首先用一个在ImageNet数据集上做了预训练的卷积神经网络模型，接着用这个模型来定义一个损失函数，表示我们对于当前图片的不满意程度(和标准答案的差异程度)。接着用反向传播来计算损失函数对于图片像素的梯度。接着保持模型不变，对图片进行梯度下降，从而生成一张新图片来降低损失(loss)。(笔者注: 模型里面的参数是不参与梯度下降的，只有输入的图片像素点会根据梯度做调整)

我们在这次作业里面主要探索三种图片生成的技巧:

1. **Saliency Maps**: Saliency maps 是一个很快的方法来说明图片中的哪些部分影响了模型对于最后那个分类label的判断。
2. **Fooling Images**: 我们可以扰乱一个输入的图片，使它对于人类来说看起来是一样的，但是会被我们的与训练的模型分错类。
3. **Class Visualization**: 我们可以合成一张图片来最大化一个特定类的打分;这可以给我们一些直观感受，来看看模型在判断图片是当前这个类的时候它在关注的是图片的哪些部分。

这个Notebook用的是TensorFlow; 我们提供了另一个内容类似的PyTorch Notebook。你只用完成这两份Notebook之一。

笔者注:

从这份作业开始，后面的notebook大小都会超过1M，对于大多数同学正常来说是没什么问题的。但是如果同学是在服务器上用了nginx做了反向代理，请务必注意nginx的客户端默认文件大小是1M，也就是说超过1M之后的文件的更改将不被保存，所以请修改nginx的配置点这里。 另外，在ipython notebook 5.0开始之后的文件大小被设置为了100兆，这个参数也是可以修改的，修改方法。

另外，同学们可以在运行的过程中用`print tensor.get_shape()`的方式来看一下tensor的形状大小是不是符合预期。

```
# 初始化，设置一些参数
from __future__ import print_function
import time, os, json
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

from cs231n.classifiers.squeezenet import SqueezeNet
from cs231n.data_utils import load_tiny_imagenet
from cs231n.image_utils import preprocess_image, deprocess_image
from cs231n.image_utils import SQUEEZENET_MEAN, SQUEEZENET_STD


%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'
```

```
# 设置使用GPU_0
# 可以实用nvidia-smi查看GPU的情况
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "0"

def get_session():
    """Create a session that dynamically allocates memory."""
    # See: https://www.tensorflow.org/tutorials/using_gpu#allowing_gpu_memory_growth
    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    session = tf.Session(config=config)
    return session

# for auto-reloading external modules
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2
```

预训练模型

对于接下来的图像生成实验，我们都会用一个预训练模型。这个模型是在ImageNet上训练的分类卷积神经网络模型。我们这里可以用任何模型，但是这次作业我们用了SqueezeNet[1],该模型效果和AlexNet差不多，但是参数和计算复杂度大大的减小了。

用SqueezeNet而不是AlexNet或者ResNet，意味着我们可以在CPU上做所有下面的图片生成实验。

我们已经把PyTorch版本的SqueezeNet模型转成了TensorFlow; 参照文件cs231n/classifiers/squeezenet.py来了解模型的结构。

为了用SqueezeNet, 首先需要下载模型参数。在目录cs231n/datasets下面，运行get_squeezenet_tf.sh，注意，如果你运行过get_assignment3_data.sh，那么SqueezeNet就已经下载了。

一旦你下载好Squeezenet模型，我们就可以用一个新的TensorFlow session来load了。 [1]Iandola et al, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5MB model size", arXiv 2016

```
tf.reset_default_graph()
sess = get_session()

SAVE_PATH = 'cs231n/datasets/squeezenet.ckpt'  #只是个前缀，并不是文件目录,是文件
if not os.path.exists(SAVE_PATH):
    raise ValueError("You need to download SqueezeNet!")
model = SqueezeNet(save_path=SAVE_PATH, sess=sess)
```

```
INFO:tensorflow:Restoring parameters from cs231n/datasets/squeezenet.ckpt
```

笔者注: 可能有同学明明下载了脚本里面的文件，并且也把文件放到对应的datasets/目录下，如果这里还是报错的话，可以考虑把if语句注释掉，看一下模型是否可以找到对应的文件并加载。

加载 ImageNet 图片

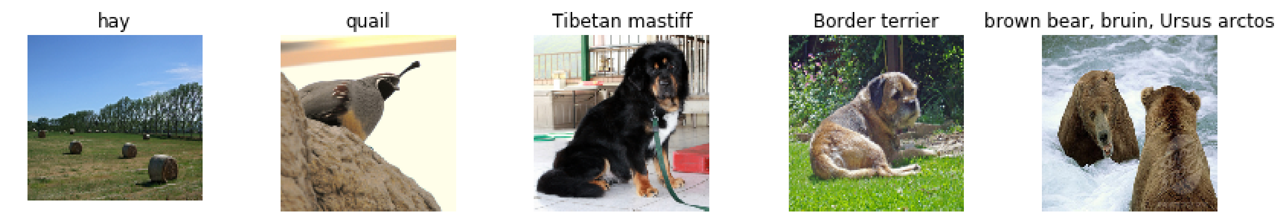
我们从 ImageNet ILSVRC 2012的分类数据集的验证集上下载了一些图片例子。

在cs231n/datasets下面运行get_imagenet_val.sh。

由于它们是从验证集里面挑选的，所以我们的预训练模型在训练期间并没有看到过这些图片。

运行下面的代码来看看这些图片和对应的label。

```python
from cs231n.data_utils import load_imagenet_val
X_raw, y, class_names = load_imagenet_val(num=5)

plt.figure(figsize=(12, 6))
for i in range(5):
    plt.subplot(1, 5, i + 1)
    plt.imshow(X_raw[i])
    plt.title(class_names[y[i]])
    plt.axis('off')
plt.gcf().tight_layout()
```



预处理图片

预训练模型的输入应该是归一化的(normalized)，所以我们要先对图片进行预处理--把图片减去均值再除以标准差。

预训练函数在: cs231n.image_utils

具体操作:

SQUEEZENET_MEAN = np.array([0.485, 0.456, 0.406], dtype=np.float32)

SQUEEZENET_STD = np.array([0.229, 0.224, 0.225], dtype=np.float32)

return (img.astype(np.float32)/255.0 - SQUEEZENET_MEAN) / SQUEEZENET_STD

这里的SQUEEZENET_MEAN和SQUEEZENET_STD都是在训练集上的预先计算好的RGB三个通道上的均值和标准差。

```python
X = np.array([preprocess_image(img) for img in X_raw])
```

Saliency Maps

用预训练模型，我们就可以根据[2]的3.1描述的那样计算saliency maps。

一张saliency map告诉了我们在图片中的每个像素点对于这张图片最后的预测得分的影响程度。为了计算它，我们要计算正确的那个类的未归一化的打分对于图片中每个像素点的梯度。如果图片的尺寸是(H,W,3),那么梯度的尺寸也应该是(H,W,3);对于图片中的每个像素点,梯度值反映了如果某个像素点的值改变一点点，分类的打分(score)会改变的程度大小。为了计算saliency map, 我们用梯度的绝对值，然后在3个channel上面求最大值，因此最后的saliency map的形状应该是(H,W)，并且所有的值都是非负数。

你会需要`model.classifier`这个包含了每个输入的打分的张量，计算梯度的时候也需要把值传给`model.image`和`model.labels`这两个placeholders。打开`cs231n/classifiers/squeezenet.py`，阅读文档并且确保你明白要怎么用这个模型。例如，你可以看看loss这个属性。

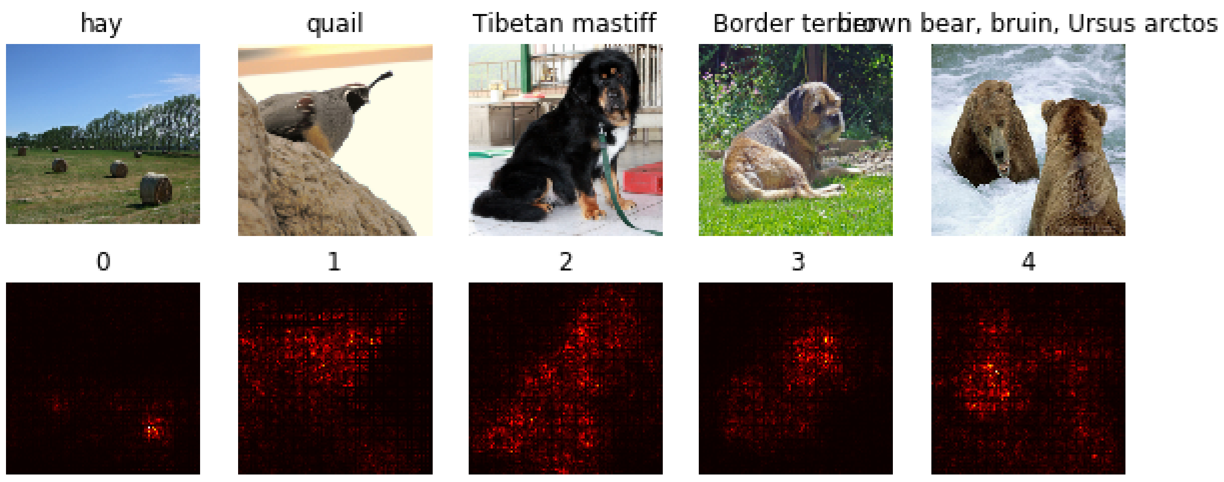
[2] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.

笔者注: 这里就是要计算我们模型最后那个打分在**正确的label上的值**对于**输入的图片**上的梯度。

```
def compute_saliency_maps(X, y, model):
    """
    Compute a class saliency map using the model for images X and labels y.

    Input:
    - X: Input images, numpy array of shape (N, H, W, 3)
    - y: Labels for X, numpy of shape (N,)
    - model: A SqueezeNet model that will be used to compute the saliency map.

    Returns:
    - saliency: A numpy array of shape (N, H, W) giving the saliency maps for the
    input images.
    """
    saliency = None
    # Compute the score of the correct class for each example.
    # This gives a Tensor with shape [N], the number of examples.
    #
    # Note: this is equivalent to scores[np.arange(N), y] we used in NumPy
    # for computing vectorized losses.

    # 计算正确的label的打分
    # model.classifier是在各个类上的打分，{batch_size, NUM_CLASSES}, 也就是logits
    # 下面这一步的运算相当于就是numpy中的 scores[np.arange(N),y]
    correct_scores = tf.gather_nd(model.classifier,
                                  tf.stack((tf.range(X.shape[0]), model.labels), axis=1))
    ###############################################################################
    # TODO: Implement this function. You should use the correct_scores to compute #
    # the loss, and tf.gradients to compute the gradient of the loss with respect #
    # to the input image stored in model.image.                                   #
    # Use the global sess variable to finally run the computation.                #
    # Note: model.image and model.labels are placeholders and must be fed values  #
    # when you call sess.run().                                                   #
    ###############################################################################
    saliency_grad = tf.gradients(correct_scores,model.image)  # 打分对于输入图像的梯度
    #print(correct_scores.shape)
    #print(model.image.shape)
    saliency = sess.run(saliency_grad, {model.image:X, model.labels:y})[0]  # 运算求值

    saliency = np.absolute(saliency) # 求绝对值
    saliency = np.amax(saliency, axis = -1) # 求三个channel上最大的值
    ##############################################################################
    #                               END OF YOUR CODE                             #
    ##############################################################################
    return saliency
```

完成上面的代码后，运行下面的代码来看看我们从ImageNet验证集上的样例图片:

```
def show_saliency_maps(X, y, mask):
    mask = np.asarray(mask)
    Xm = X[mask]
    ym = y[mask]

    saliency = compute_saliency_maps(Xm, ym, model)

    for i in range(mask.size):
        plt.subplot(2, mask.size, i + 1)
        plt.imshow(deprocess_image(Xm[i]))
        plt.axis('off')
        plt.title(class_names[ym[i]])
        plt.subplot(2, mask.size, mask.size + i + 1)
        plt.title(mask[i])
        plt.imshow(saliency[i], cmap=plt.cm.hot)
        plt.axis('off')
        plt.gcf().set_size_inches(10, 4)
    plt.show()

mask = np.arange(5)
show_saliency_maps(X, y, mask)
```



Fooling Images

我们也可以用图像梯度来生成一些"fooling images"，正如[3]中讨论的那样。 给定了一张图片和一个目标的类，我们可以在图片上做梯度上升来最大化目标类的分数，直到神经网络把这个图片预测为目标类位置。 实现下面的函数来生成fooling images。

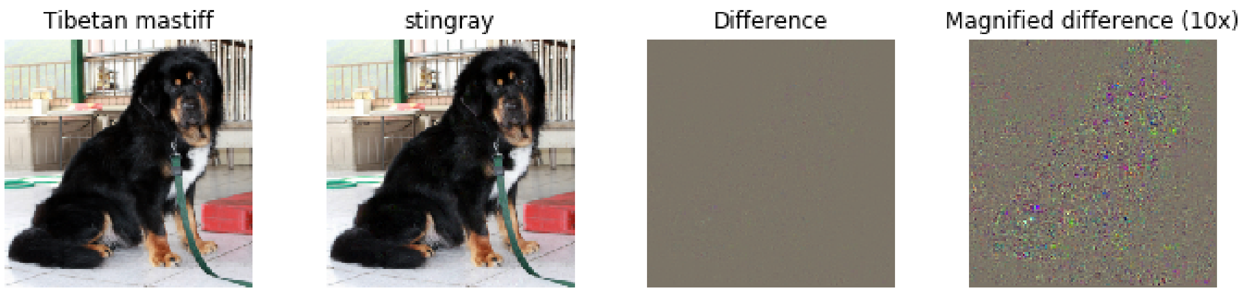
[3] Szegedy et al, "Intriguing properties of neural networks", ICLR 2014

```
from numpy import linalg as LA
def make_fooling_image(X, target_y, model):
    """
    Generate a fooling image that is close to X, but that the model classifies
    as target_y.

    Inputs:
    - X: Input image, of shape (1, 224, 224, 3)
    - target_y: An integer in the range [0, 1000)
    - model: Pretrained SqueezeNet model

    Returns:
    - X_fooling: An image that is close to X, but that is classifed as target_y
    by the model.
    """
    X_fooling = X.copy()
    learning_rate = 1
    ##############################################################################
    # TODO: Generate a fooling image X_fooling that the model will classify as   #
    # the class target_y. Use gradient ascent on the target class score, using   #
    # the model.classifier Tensor to get the class scores for the model.image.   #
    # When computing an update step, first normalize the gradient:               #
    #   dX = learning_rate * g / ||g||_2                                         #
    #                                                                            #
    # You should write a training loop                                           #
    #                                                                            #
    # HINT: For most examples, you should be able to generate a fooling image    #
    # in fewer than 100 iterations of gradient ascent.                           #
    # You can print your progress over iterations to check your algorithm.       #
    ##############################################################################
    for i in range(100):
        target_score = tf.gather_nd(model.classifier,
                                    tf.stack((tf.range(X.shape[0]), model.labels), axis=1))  # 当
前label下面的对应的打分值
        grad_tensor = tf.gradients(target_score,model.image) # 目标类对于模型图片求梯度
        grad,predict_logit = sess.run([grad_tensor,model.classifier],{model.image:X_fooling,
model.labels:[target_y]}) #给模型目标类的label
        cur_predict_label = np.argmax(predict_logit, axis = 1)[0]  # 当前模型预测的label
        if cur_predict_label == target_y:  # 判断预测label是否和目标类一致
            print("The labels are the same.")
            #print(cur_predict_label)
            #print(target_y)
            return X_fooling
        grad = grad[0]
        dx = learning_rate * grad/ LA.norm(grad)


        X_fooling = X_fooling + dx  # 梯度上述
    ##############################################################################
    #                               END OF YOUR CODE                             #
    ##############################################################################
    return X_fooling
```

运行下面的代码生成fooling image。可以把 `idx` 变量进行修改来查看其它图片。

```
idx = 2
Xi = X[idx][None]
target_y = 6
X_fooling = make_fooling_image(Xi, target_y, model)

# Make sure that X_fooling is classified as y_target
scores = sess.run(model.classifier, {model.image: X_fooling})
assert scores[0].argmax() == target_y, 'The network is not fooled!'

# Show original image, fooling image, and difference
orig_img = deprocess_image(Xi[0])
fool_img = deprocess_image(X_fooling[0])
# Rescale
plt.subplot(1, 4, 1)
plt.imshow(orig_img)
plt.axis('off')
plt.title(class_names[y[idx]])
plt.subplot(1, 4, 2)
plt.imshow(fool_img)
plt.title(class_names[target_y])
plt.axis('off')
plt.subplot(1, 4, 3)
plt.title('Difference')
plt.imshow(deprocess_image((Xi-X_fooling)[0]))
plt.axis('off')
plt.subplot(1, 4, 4)
plt.title('Magnified difference (10x)')
plt.imshow(deprocess_image(10 * (Xi-X_fooling)[0]))
plt.axis('off')
plt.gcf().tight_layout()
```

```
The labels are the same.
```



类别可视化

通过产生一个随机噪声的图片，然后在目标类上做梯度上升，我们就可以生成一张模型会认为是目标类的图片了。这个想法第一次是在[2]中发表的；[3]又拓展了这个想法通过加入一些正则的技巧来提升生成图片的质量。

具体来说，假设I是一张图片，y是目标类. 假设$s_y(I)$是卷积网络在图片I是目标类y上面的打分; 注意这些都是未归一化的打分, 并不是类的概率.我们希望通过解决下面这个公式来可以生成一张图片I^* 能够在目标类y上得分高。

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

其中R是一个正则项(注意$R(I)$在argmax中的符号: 我们希望减小这个正则项)。我们可以用梯度上升来解决这个优化问题, 计算对于生成图片上的梯度. 我们用明确的L2正则项:

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

以及 [3]中建议的隐式的正则向，通过周期性的模糊图片。我们可以对生成图片做梯度上升来解决这个问题。

完成下面的`create_class_visualization`函数。

[2] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.

[3] Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML 2015 Deep Learning Workshop

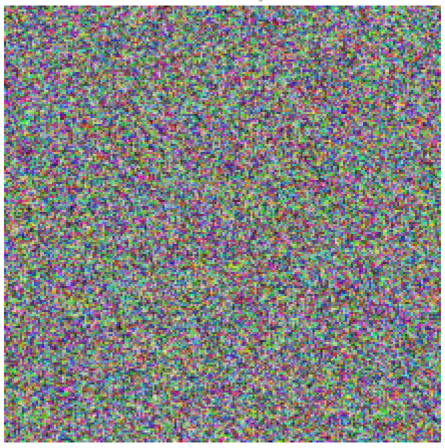
```python
from scipy.ndimage.filters import gaussian_filter1d
def blur_image(X, sigma=1):
    X = gaussian_filter1d(X, sigma, axis=1)
    X = gaussian_filter1d(X, sigma, axis=2)
    return X
```

```python
def create_class_visualization(target_y, model, **kwargs):
    """
    Generate an image to maximize the score of target_y under a pretrained model.

    Inputs:
    - target_y: Integer in the range [0, 1000) giving the index of the class
    - model: A pretrained CNN that will be used to generate the image

    Keyword arguments:
    - l2_reg: Strength of L2 regularization on the image
    - learning_rate: How big of a step to take
    - num_iterations: How many iterations to use
    - blur_every: How often to blur the image as an implicit regularizer
    - max_jitter: How much to gjitter the image as an implicit regularizer
    - show_every: How often to show the intermediate result
    """
    l2_reg = kwargs.pop('l2_reg', 1e-3)
    learning_rate = kwargs.pop('learning_rate', 25)
    num_iterations = kwargs.pop('num_iterations', 100)

    blur_every = kwargs.pop('blur_every', 10)
    max_jitter = kwargs.pop('max_jitter', 16)
    show_every = kwargs.pop('show_every', 25)

    X = 255 * np.random.rand(224, 224, 3)
    X = preprocess_image(X)[None]

    ########################################################################
    # TODO: Compute the loss and the gradient of the loss with respect to  #
    # the input image, model.image. We compute these outside the loop so   #
    # that we don't have to recompute the gradient graph at each iteration #
    #                                                                      #
    # Note: loss and grad should be TensorFlow Tensors, not numpy arrays!  #
    #                                                                      #
    # The loss is the score for the target label, target_y. You should     #
    # use model.classifier to get the scores, and tf.gradients to compute  #
    # gradients. Don't forget the (subtracted) L2 regularization term!     #
    ########################################################################
    loss = None # scalar loss
    grad = None # gradient of loss with respect to model.image, same size as model.image

    loss = tf.gather_nd(model.classifier,
                              tf.stack((tf.range(X.shape[0]), model.labels), axis=1))
    l2_reg_tensor = tf.constant(l2_reg)
    loss = tf.subtract(loss, tf.multiply(l2_reg_tensor,tf.nn.l2_normalize(model.image,dim=
[0,1,2,3])))
    grad = tf.gradients(loss,model.image)

    ############################################################################
    #                             END OF YOUR CODE                             #
    ############################################################################


    for t in range(num_iterations):
        # Randomly jitter the image a bit; this gives slightly nicer results
        ox, oy = np.random.randint(-max_jitter, max_jitter+1, 2)
        Xi = X.copy()
        X = np.roll(np.roll(X, ox, 1), oy, 2)

        ########################################################################
        # TODO: Use sess to compute the value of the gradient of the score for #
        # class target_y with respect to the pixels of the image, and make a   #
        # gradient step on the image using the learning rate. You should use   #
        # the grad variable you defined above.                                 #
        #                                                                      #
        # Be very careful about the signs of elements in your code.            #
        ########################################################################
        #print(X.shape)
        #print(model.image.get_shape())
        gradient = sess.run([grad], {model.image:X, model.labels:[target_y]})[0]
        gradient = gradient[0]
        dx = learning_rate * gradient/ LA.norm(gradient)
        X = X + dx

        #print("after gradient",X.shape)
        ############################################################################
        #                             END OF YOUR CODE                             #
        ############################################################################

        # Undo the jitter
        X = np.roll(np.roll(X, -ox, 1), -oy, 2)

        # As a regularizer, clip and periodically blur
        X = np.clip(X, -SQUEEZENET_MEAN/SQUEEZENET_STD, (1.0 - SQUEEZENET_MEAN)/SQUEEZENET_STD)
        if t % blur_every == 0:
            X = blur_image(X, sigma=0.5)

        # Periodically show the image
        if t == 0 or (t + 1) % show_every == 0 or t == num_iterations - 1:
            plt.imshow(deprocess_image(X[0]))
            class_name = class_names[target_y]
            plt.title('%s\nIteration %d / %d' % (class_name, t + 1, num_iterations))
            plt.gcf().set_size_inches(4, 4)
            plt.axis('off')
            plt.show()
    return X
```

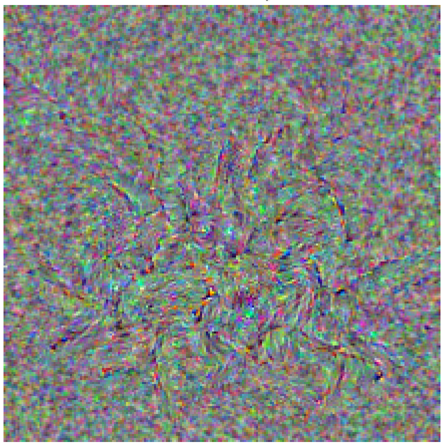
一旦你实现了上面的功能，运行下面的代码来生成一张狼蛛的图片

```
target_y = 76 # Tarantula
out = create_class_visualization(target_y, model)
```

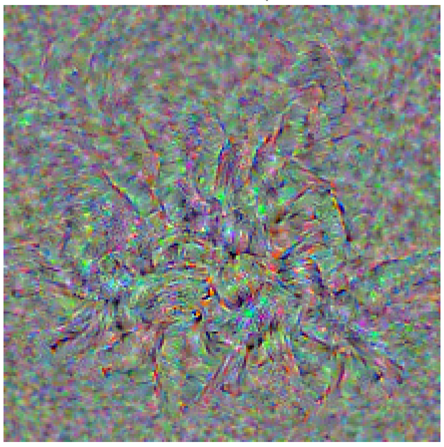
tarantula
Iteration 1 / 100



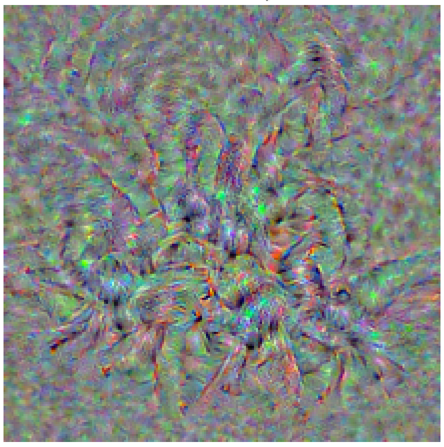
tarantula
Iteration 25 / 100



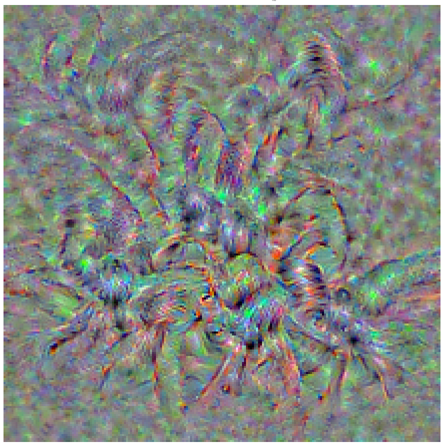
tarantula
Iteration 50 / 100



tarantula
Iteration 75 / 100



tarantula
Iteration 100 / 100

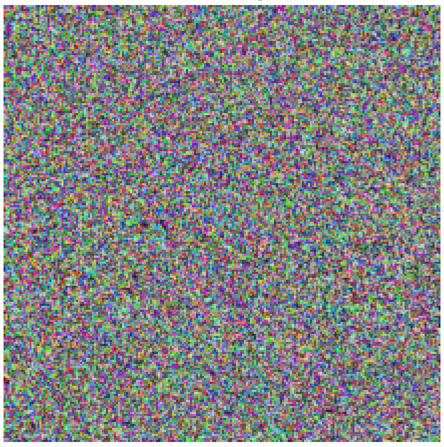


试试在其它类上做一些可视化，你可以用各种超参数去尝试提高图片的质量，但不是必须的任务。

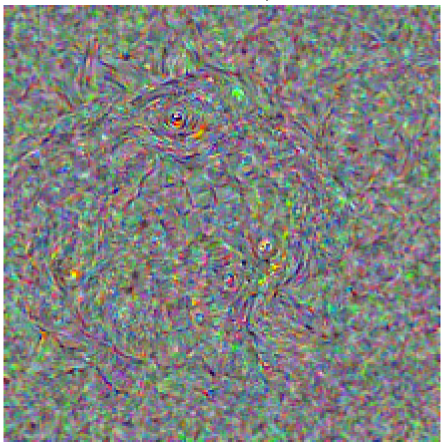
```
# target_y = 78 # Tick
# target_y = 187 # Yorkshire Terrier
# target_y = 683 # Oboe
# target_y = 366 # Gorilla
# target_y = 604 # Hourglass
target_y = np.random.randint(1000)
print(class_names[target_y])
X = create_class_visualization(target_y, model)
```

```
box turtle, box tortoise
```

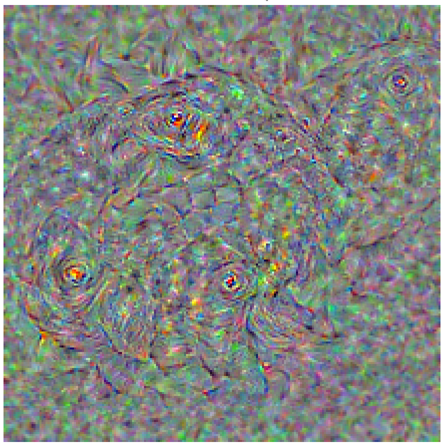
box turtle, box tortoise
Iteration 1 / 100



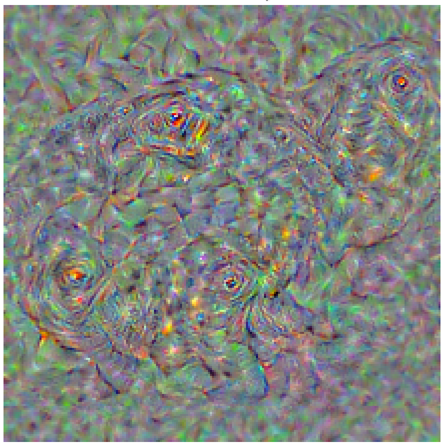
box turtle, box tortoise
Iteration 25 / 100



box turtle, box tortoise
Iteration 50 / 100



box turtle, box tortoise
Iteration 75 / 100



box turtle, box tortoise
Iteration 100 / 100