Network Visualization (PyTorch)

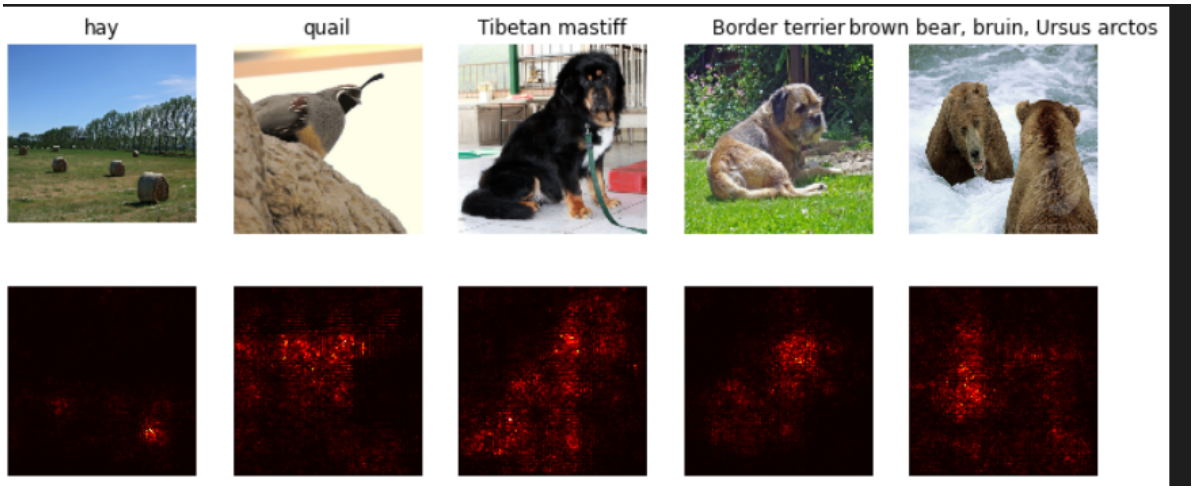
Saliency Maps

通过梯度大小表示某一个部分对于结果的影响大小，表示网络主要观察的是哪个部分，从而实现可视化网络。

```
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; Tensor of shape (N, 3, H, W)
    - y: Labels for X; LongTensor of shape (N,)
    - model: A pretrained CNN that will be used to compute the saliency map.

    Returns:
    - saliency: A Tensor of shape (N, H, W) giving the saliency maps for the
input
    images.
    """
    # Make sure the model is in "test" mode
    model.eval()

    # Make input tensor require gradient
    X.requires_grad_()

    saliency = None

 ##############################################################################
    # TODO: Implement this function. Perform a forward and backward pass through
#
    # the model to compute the gradient of the correct class score with respect
#
    # to each input image. You first want to compute the loss over the correct
#
    # scores (we'll combine losses across a batch by summing), and then compute
#
    # the gradients with a backward pass.
#

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

    scores=model(X)#前向操作
    loss=scores.gather(1, y.view(-1, 1)).squeeze().sum()#combine losses across a
batch by summing计算总的loss
    loss.backward()#反向传播
    saliency=torch.max(torch.abs(X.grad),1)[0]#max（X，1）降维1dim，【0】输出tensor
否则是对象
    # print(saliency)
    # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

 ##############################################################################
```

```
    #                             END OF YOUR CODE
#

 ##############################################################################
    return saliency
```

结果：



亮的地方表示梯度大，对应对于网络影响大的部分，说明网络focus这一部分，对于网络有解释性。

能否使用这里的梯度更新？不能，因为这里可视化了最大梯度，而原图的梯度更新需要三个维度。

Fooling Images

网络对于一张图片产生较小的差异就能使网络将图片完全分成另一个类别。

```
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; Tensor of shape (1, 3, 224, 224)
    - target_y: An integer in the range [0, 1000)
    - model: A pretrained CNN

    Returns:
    - X_fooling: An image that is close to X, but that is classifed as target_y
    by the model.
    """
    # Initialize our fooling image to the input image, and make it require
gradient
    X_fooling = X.clone()
    X_fooling = X_fooling.requires_grad_()

    learning_rate = 1

 ##############################################################################
    # TODO: Generate a fooling image X_fooling that the model will classify as
#
    # the class target_y. You should perform gradient ascent on the score of the
#
    # target class, stopping when the model is fooled.
#
```

```
    # 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.
#

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

    for epoch in range(100):
        print('epoch:',epoch)
        scores=model(X_fooling)#前向传播
        # print(scores)
        if scores.argmax(dim=1)==target_y:#网络将图片分成target--y类
            break
        #Score for the target class直接通过分数计算损失也可以是别的
        loss=scores[0][target_y]
        loss.backward()
        dx=learning_rate*X_fooling.grad.data/X_fooling.grad.data.norm()

        X_fooling.data+=dx #通过gradient asent更新产生target--y的图片对应有具体的算法
        ## 梯度更新清零，否则梯度会进行累积.
        X_fooling.grad.data.zero_()

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

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

 ##############################################################################
    return X_fooling
```

效果:

很微小的差异就能使网络分成完全不同的类别。

Class visualization

可视化网络究竟用什么pattern匹配图片，随机初始化一张图片，通过对于X梯度上升产生某种类别的图片，这种方法得出网络究竟在用什么模式匹配.

原理

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

- $s_y(I)$图像在y类的得分
- 目标产生图片使在y类得分最大
- R隐式正则化使图像变平滑

通过梯度上升解上述式子，因为最终是要最大化一个函数，最小化对应梯度下降，最大化相当于要梯度上升

```
def create_class_visualization(target_y, model, dtype, **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
    - dtype: Torch datatype to use for computations

    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
    """
    model.type(dtype)
    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)

    # Randomly initialize the image as a PyTorch Tensor, and make it requires
gradient.
    img = torch.randn(1, 3, 224, 224).mul_(1.0).type(dtype).requires_grad_()

    for t in range(num_iterations):
        # Randomly jitter the image a bit; this gives slightly nicer results
        ox, oy = random.randint(0, max_jitter), random.randint(0, max_jitter)
        img.data.copy_(jitter(img.data, ox, oy))

        ########################################################################
        # TODO: Use the model to compute the gradient of the score for the     #
        # class target_y with respect to the pixels of the image, and make a   #
        # gradient step on the image using the learning rate. Don't forget the #
        # L2 regularization term!                                              #
        # Be very careful about the signs of elements in your code.            #
        ########################################################################
```

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

        scores=model(img)
        loss=scores[0][target_y]+l2_reg*torch.sum(img**2)#直接用分数做损失
        loss.backward()
        img.data+=learning_rate* img.grad.data/img.grad.data.norm()#梯度上升求解
        img.grad.zero_() #pytorch步骤把梯度归0


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

        # Undo the random jitter
        img.data.copy_(jitter(img.data, -ox, -oy))

        # As regularizer, clamp and periodically blur the image
        for c in range(3):
            lo = float(-SQUEEZENET_MEAN[c] / SQUEEZENET_STD[c])
            hi = float((1.0 - SQUEEZENET_MEAN[c]) / SQUEEZENET_STD[c])
            img.data[:, c].clamp_(min=lo, max=hi)
        if t % blur_every == 0:
            blur_image(img.data, sigma=0.5)

        # Periodically show the image
        if t == 0 or (t + 1) % show_every == 0 or t == num_iterations - 1:
            plt.imshow(deprocess(img.data.clone().cpu()))
            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 deprocess(img.data.cpu())
```

日本狗效果图片（多个狗头）：