可视化_CNN

<Excerpt in index | 首页摘要>
Grad-CAM: VisualExplanationsfromDeepNetworksviaGradient-basedLocalization
Guided-Backpropagation, Striving for Simplicity: The All Convolutional Net
Deconvolution：Visualizing and Understanding Convolutional Networks
从本质上说，反卷积和导向反向传播的基础都是反向传播，二者唯一的区别在于反向传播过程中经过ReLU层时对梯度的不同处理策略。
1、CNN 2、可解释性 3、反卷积(Deconvolution) 4、导向反向传播(Guided-backpropagation)
Github：Tensorflow_Grad_CAM_plus_plus
Grad-CAM-tensorflow
Paper:Grad-CAM++
Guided-Backpropagation
Visualizing and Understanding Convolutional Networks

<The rest of contents | 余下全文>

Visualizing CNN

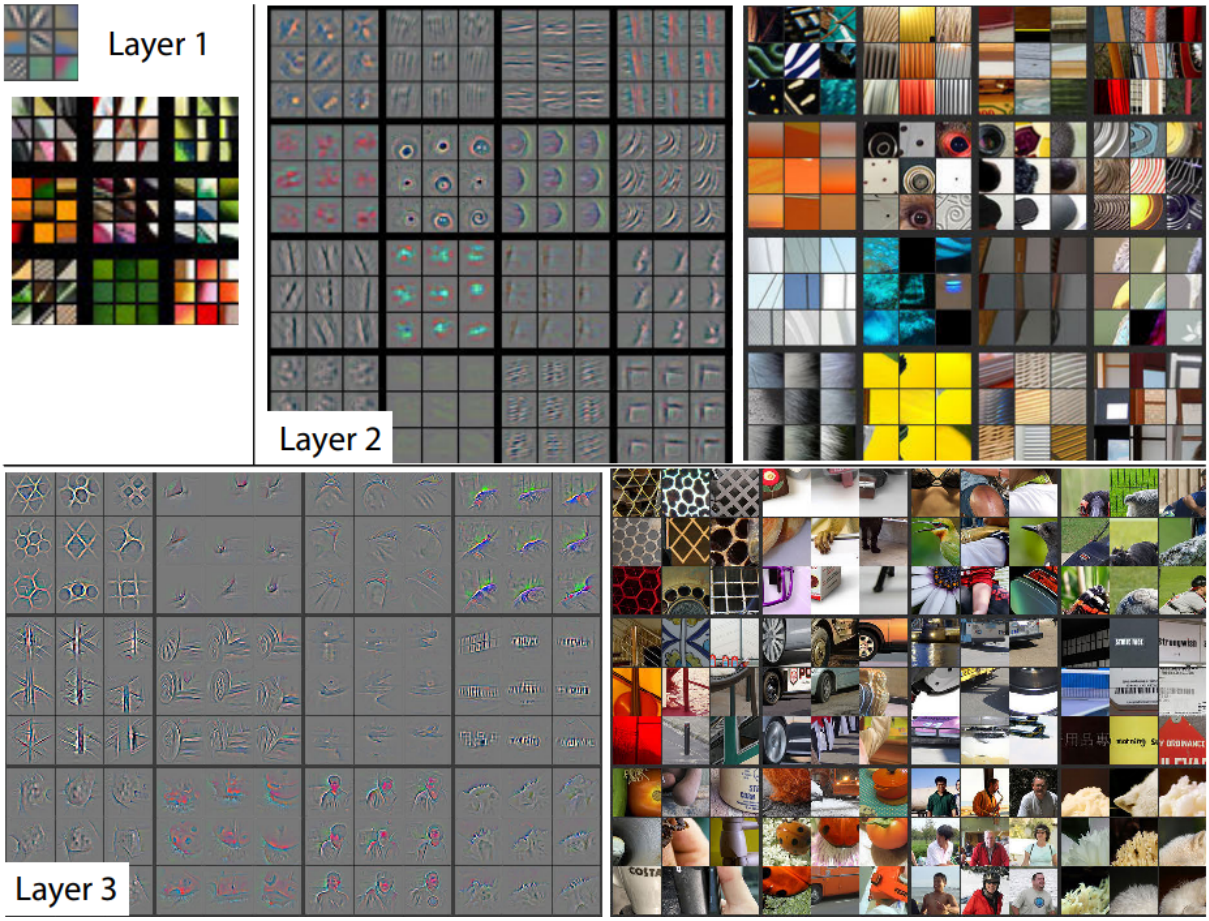
Paper:Visualizing and Understanding Convolutional Networks

反卷积实现特征可视化：反卷积可视化以各层得到的特征图作为输入，进行反卷积，得到反卷积结果，用以验证显示各层提取到的特征图。Eg：假如你想要查看`Alexnet`的`conv5`提取到了什么东西，我们就用conv5的特征图后面接一个反卷积网络，然后通过：反池化、反激活、反卷积，这样的一个过程，**把本来一张13*13大小的特征图(conv5大小为13*13)，放大回去，最后得到一张与原始输入图片一样大小的图片(227*227)。**

具体的**反池化、反卷积、反激活**操作参考paperVisualizing and Understanding Convolutional Networks和blog

Result

通过CNN学习后，**背景部位的激活度基本很少**，我们通过可视化就可以看到我们提取到的特征忽视了背景，而是把关键的信息给提取出来了。**从layer 1、layer 2学习到的特征基本上是颜色、边缘、轮廓等低层特征**；**layer 3则开始稍微变得复杂**，学习到的是纹理特征，比如上面的一些网格纹理；**layer 4学习到的则是比较有区别性的特征**；**layer 5学习到的则是完整的**，具有辨别性关键特征。



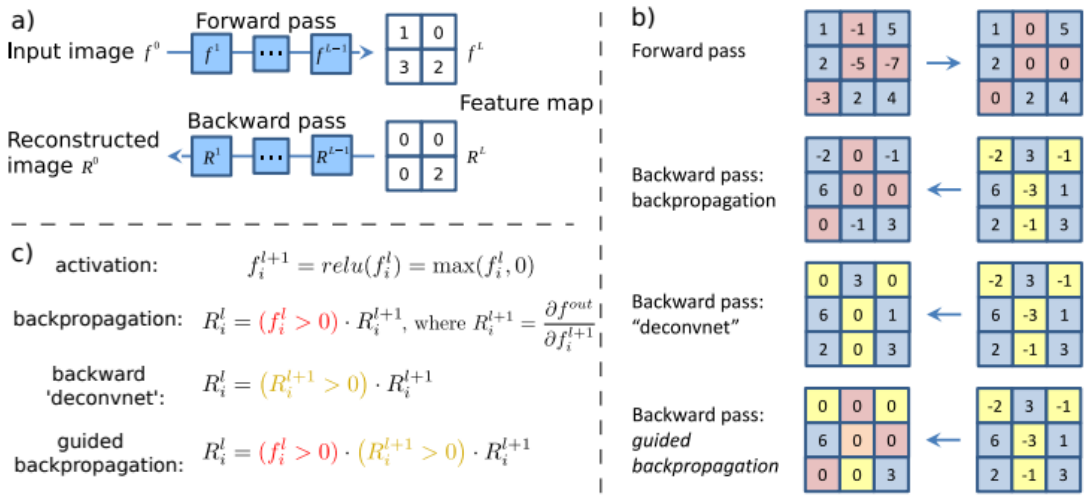
Guided-Backpropagation

Paper:Guided-Backpropagation

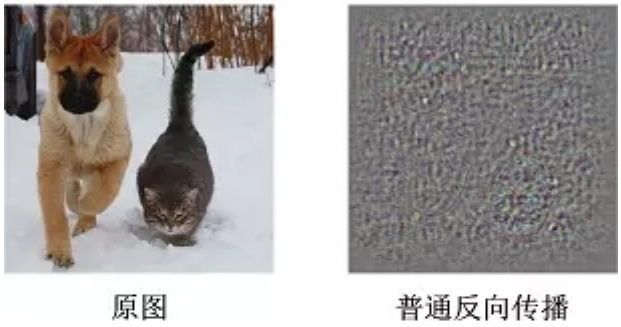
Blog:反卷积和导向反向传播

· `反卷积(Deconvolution)` 和 `导向反向传播(Guided-backpropagation)`

从本质上说，反卷积和导向反向传播的基础都是反向传播，其实说白了就是对输入进行求导，二者唯一的区别在于反向传播过程中· `经过ReLU层时对梯度的不同处理策略。`



区别看起来没有非常微小，但是在最终的效果上却有很大差别。如下图所示：



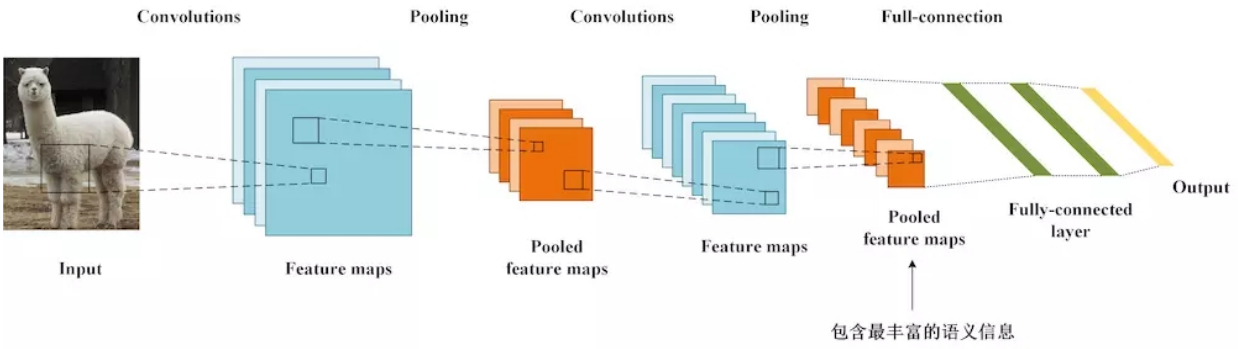
使用普通的反向传播得到的图像噪声较多，基本看不出模型的学到了什么东西。使用反卷积可以大概看清楚猫和狗的轮廓，但是有大量噪声在物体以外的位置上。导向反向传播基本上没有噪声，特征很明显的集中猫和狗的身体部位上。

借助反卷积和导向反向传播的结果并不能拿来解释分类的结果，因为它们对类别并不敏感，直接把所有能提取的特征都展示出来了。在刚才的图片中，模型给出的分类结果是猫，但是通过反卷积和导向反向传播展示出来的结果却同时包括了狗的轮廓。换句话说，**我们并不知道模型到底是通过哪块区域判断出当前图片是一只猫的。**

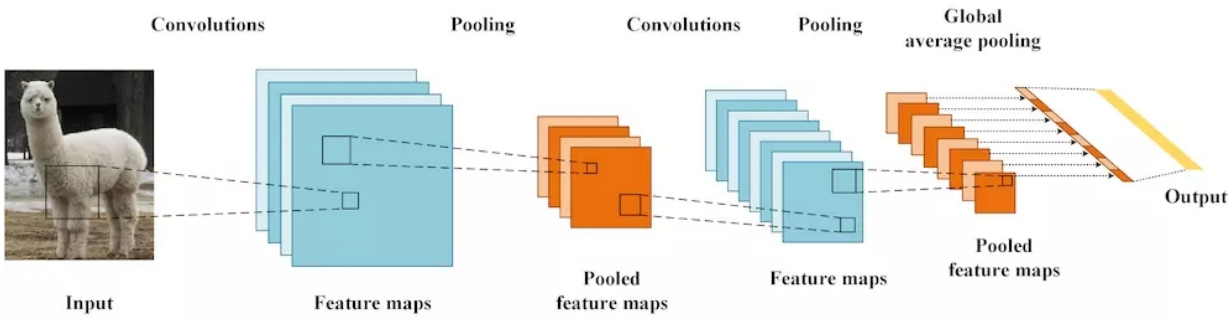
CAM

`CAM(Class Activation Mapping)`产生的CAM图以热力图（Saliency Map）的形式展示它的决策依据.

对一个深层的卷积神经网络而言，通过多次卷积和池化以后，它的**最后一层卷积层**包含了最丰富的空间和语义信息，再往下就是全连接层和softmax层了，其中所包含的信息都是人类难以理解的，很难以可视化的方式展示出来。

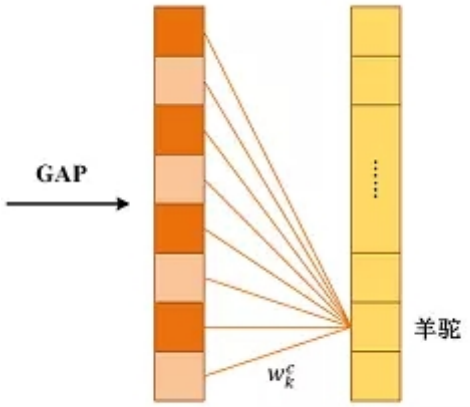


CAM借鉴了很著名的论文Network in Network中的思路，利用`GAP(Global Average Pooling)`替换掉了**全连接层**。可以把GAP视为一个特殊的average pool层，只不过其**pool size和整个特征图一样大**，其实说白了就是求每张特征图所有像素的均值。



最后一层卷积层**强制生成了和目标类别数量一致的特征图**，经过GAP以后再通过softmax层得到结果，这样做就给每个特征图赋予了很明确的意义，也就是`categories confidence maps`。

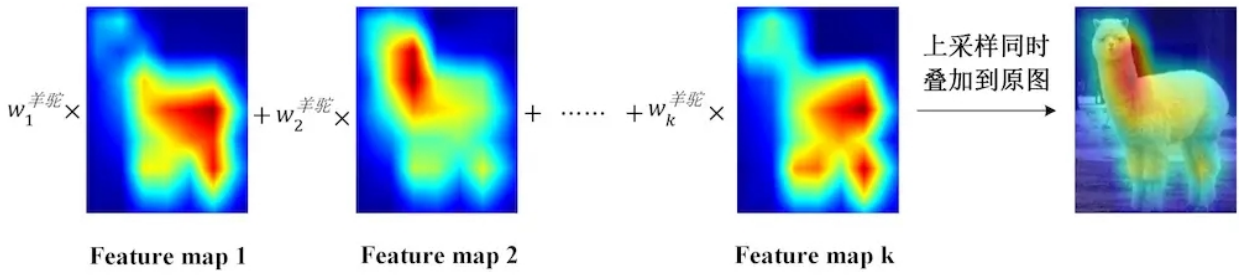
经过GAP之后与输出层的连接关系(暂不考虑softmax层)，实质上也是就是个全连接层，只不过没有了偏置项，如图所示：



w为权重，k表示对应的特征图，c为对应的类别

从图中可以看到，经过GAP之后，我们得到了最后一个卷积层每个特征图的均值，通过加权和得到输出(实际中是softmax层的输入)。

我们如何得到一个用于解释分类结果的热力图呢？比如说我们要解释为什么分类的结果是羊驼，我们把羊驼这个类别对应的**所有W取出来，求出它们与自己对应的特征图的加权和即可**。由于这个结果的大小和特征图是一致的，我们需要**对它进行上采样，叠加到原图上去。**



这样，CAM以热力图的形式告诉了我们，模型是**重点通过哪些像素确定这个图片是羊驼了。**

Grad-CAM

1、Introduction

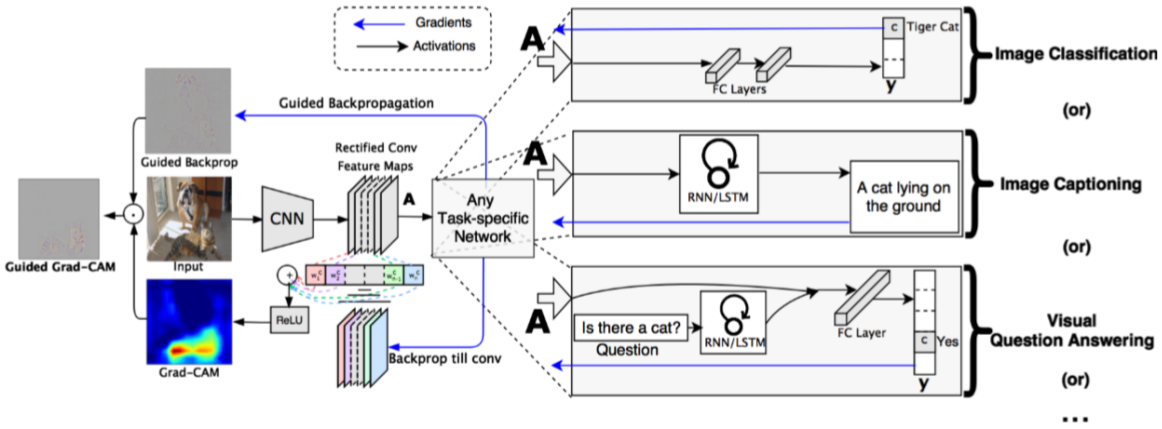
Grad-CAM的基本思路和CAM是一致的，也是通过得到每对特征图对应的权重，最后求一个加权和。但是它与CAM的主要区别在于求权重w_k^c的过程。CAM通过替换全连接层为GAP层，重新训练得到权重，而Grad-CAM另辟蹊径，**用梯度的全局平均来计算权重**。事实上，经过严格的数学推导，Grad-CAM与CAM计算出来的权重是等价的。为了和CAM的权重做区分，定义Grad-CAM中第k个特征图对类别c的权重为α_k^c，可通过下面的公式计算：

$$\alpha_k^c = \frac{1}{Z}\sum_i\sum_j\frac{\partial y^c}{\partial A_{ij}^k}$$

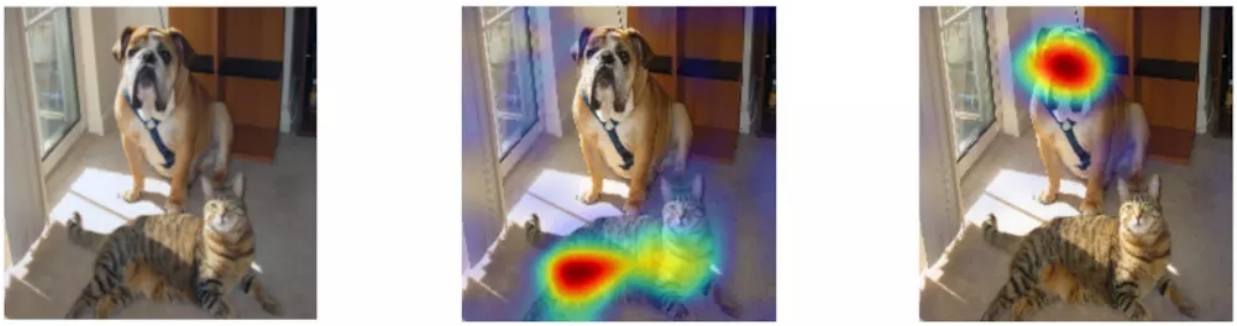
其中，Z为特征图的像素个数，y^c是对应类别c的分数（在代码中一般用`logits`表示，是输入`softmax`层之前的值），A_{ij}^k表示第k个特征图中，(i,j)位置处的像素值。求得类别对所有特征图的权重后，求其加权和就可以得到热力图。

$$L_{Grad-CAM}^c = ReLU(\sum_k \alpha_k^c A^k)$$

Grad-CAM的整体结构如下图所示：



注意这里和CAM的另一个区别是，`Grad-CAM`对最终的加权和加了一个ReLU，加这么一层ReLU的原因在于我们只关心**对类别c有正影响的那些像素点**，如果不加ReLU层，最终可能会带入一些属于其它类别的像素，从而影响解释的效果。使用Grad-CAM对分类结果进行解释的效果如下图所示：



原图　　分类为猫的依据　　分类为狗的依据

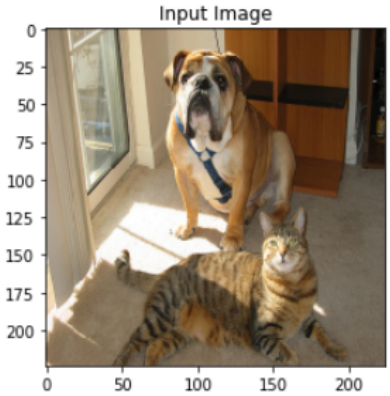
- 1.Generate visual explanations from any CNN-based network `without requiring architectural changes or re-training.`
- 2.Help untrained users successfully **discern a 'stronger' network from a 'weaker' one**, even when both make identical predictions

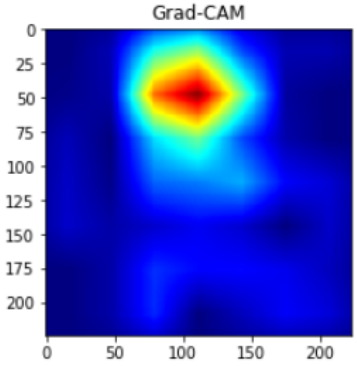
2、Code

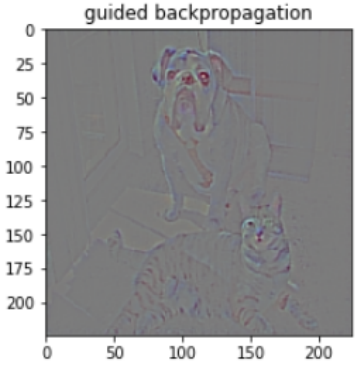
基于**Github**:Grad-CAM-tensorflow

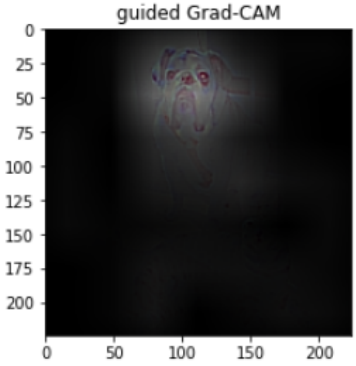
```
cost = (-1) * tf.reduce_sum(tf.multiply(labels, tf.log(prob)), axis=1)
 #用于Guided backpropagtion

# Get last convolutional layer gradient for generating gradCAM visualizat
ion
target_conv_layer = end_points['model/block4/unit_3/bottleneck_v2/conv3']

# gradient for partial linearization. We only care about target visualiza
tion class.
y_c = tf.reduce_sum(tf.multiply(net, labels), axis=1)

target_conv_layer_grad = tf.gradients(y_c, target_conv_layer)[0]

# Guided backpropagtion back to input layer
gb_grad = tf.gradients(cost, images)[0]          #
```

3、Result


Input Image


Grad-CAM


guided backpropagation


guided Grad-CAM

Grad-CAM++