RoI Proposal

Layer 10　softmax

$$\frac{\exp\left(x_i - \max_{i=1,\ldots,1000}\left(x_i\right)\right)}{\sum_{i=1}^{1000}\exp\left(x_i - \max_{i=1,\ldots,1000}\left(x_i\right)\right)}$$

输入为 reshape 后的 Layer 9_1 结果，即 126*14*2，在这里，图片变成两维的，我理解的是每两个对应位置上的特征点进行 softmax，即 k=2,首先经过 1 次比较，在经过 2 次减法，之后经过 2 次的 ej 运算，再进行 1 次加法，最后 1 次除法，二总共有 126*14 个位置，因此具有 1764 次比较，在经过 3528 次减法，之后经过 3528 次的 ej 运算，再进行 1764 次加法，最后 1764 次除法

在进行玩这一步之后，还要对其进行 Reshape，其参数为{ dim: 0 dim: 18 dim: -1 dim: 0 }，即又变成了 14*14*18

Layer 11　Proposals

输入为 Layer 10、Layer 9_2 和原图（224*224*3），看一下程序这个过程到底怎么实现的，打开 proposal_layer.py，在 set_up 中，调用了 generate_anchors()。

在这里先为自己普及一下生成 anchor 的函数实现原理,追随源码(generate_anchors.py)：

```
def generate_anchors(base_size=16, ratios=[0.5, 1, 2],
                     scales=2**np.arange(3, 6)):
    """
    Generate anchor (reference) windows by enumerating aspect ratios X
    scales wrt a reference (0, 0, 15, 15) window.
    """

    base_anchor = np.array([1, 1, base_size, base_size]) - 1
    ratio_anchors = _ratio_enum(base_anchor, ratios)
    anchors = np.vstack([_scale_enum(ratio_anchors[i, :], scales)
                         for i in xrange(ratio_anchors.shape[0])])
    return anchors
```

这个函数就是生成九个 anchors 的函数，首先有一个 base_anchor 坐标为[0，0，15，15]，因为电脑是从 0 开始计数的，其实是[1，1，16，16]，先调用_ratio_enum

```
def _ratio_enum(anchor, ratios):
    """
    Enumerate a set of anchors for each aspect ratio wrt an anchor.
    """

    w, h, x_ctr, y_ctr = _whctrs(anchor)
    size = w * h
    size_ratios = size / ratios
    ws = np.round(np.sqrt(size_ratios))
    hs = np.round(ws * ratios)
    anchors = _mkanchors(ws, hs, x_ctr, y_ctr)
    return anchors
```

在这个函数里先调用了_whctrs，作用是得到 anchor 的四个参数，宽度 w=16，高度 h=16，中心点坐标 x=7.5,y=7.5，之后做了一系列数学计算，最终结果为 ws=[23,16,11]，hs=[12,16,22]，调用_mkanchors

```
def _mkanchors(ws, hs, x_ctr, y_ctr):
    """
    Given a vector of widths (ws) and heights (hs) around a center
    (x_ctr, y_ctr), output a set of anchors (windows).
    """

    ws = ws[:, np.newaxis]
    hs = hs[:, np.newaxis]
    anchors = np.hstack((x_ctr - 0.5 * (ws - 1),
                         y_ctr - 0.5 * (hs - 1),
                         x_ctr + 0.5 * (ws - 1),
                         y_ctr + 0.5 * (hs - 1)))
    return anchors
```

在这个函数的里面有个 np.newaxis，是增加数据的维数的意思，原来 ws,hs 均为一组数据（只有一个维度），之后变成一个 3 行，1 列（虽然是 1，但是也是个维度）的二维数据，即变成了 ws=｛[23],[16],[11]｝，hs={[12],[16],[22]}，最后一个函数不想细说，总之就是变成了$\begin{bmatrix} -3.5 & 2 & 18.5 & 13 \\ 0 & 0 & 15 & 15 \\ 2.5 & -3 & 12.5 & 18 \end{bmatrix}$，即 ratio_anchors，我们在回到最基本的那个函数，接下来又调用了一个函数

```
anchors = np.vstack([_scale_enum(ratio_anchors[i, :], scales)
                     for i in xrange(ratio_anchors.shape[0])])
```

在这里 ratio_anchors.shape[0]指的是 3 行 4 列的 3，也就是一行一行的输送给_scale_enum 函数

```
def _scale_enum(anchor, scales):
    """
    Enumerate a set of anchors for each scale wrt an anchor.
    """

    w, h, x_ctr, y_ctr = _whctrs(anchor)
    ws = w * scales
    hs = h * scales
    anchors = _mkanchors(ws, hs, x_ctr, y_ctr)
    return anchors
```

由最开始的那个函数的参数可知，scales 为 2 的 3，4，5 次方即[8,16,32]。首先，先得到四个参数（w, h, x_ctr, y_ctr）=$\begin{bmatrix} 23 & 12 & 7.5 & 7.5 \\ 16 & 16 & 7.5 & 7.5 \\ 11 & 22 & 7.5 & 7.5 \end{bmatrix}$所以 ws=[184,368,736],[128,256,512],[88,176,352], hs=[96,192,384],[128,256,512], [176,352,704], 这是 9 组对应的宽和高，其实每次只能得到三组，我是直接把循环三次的结果写了出来，将这 9 组的数据送到_mkanchors 之后得到 9 个 anchors，分别为：

-84.0	-40.0	99	55
-176.0	-88.0	191	103
-360.0	-184.0	375	199
-56.0	-56.0	71	71
-120.0	-120.0	135	135
-248.0	-248.0	263	263
-36.0	-80.0	51	95
-80.0	-168.0	95	183
-168.0	-344.0	183	359

转换成我们需要的四个参数分别为：

ratio = 0.5

(184.0, 96.0, 7.5, 7.5)

(368.0, 192.0, 7.5, 7.5)

(736.0, 384.0, 7.5, 7.5)

ratio = 1.0

(128.0, 128.0, 7.5, 7.5)

(256.0, 256.0, 7.5, 7.5)

(512.0, 512.0, 7.5, 7.5)

ratio = 2.0

(88.0, 176.0, 7.5, 7.5)

(176.0, 352.0, 7.5, 7.5)

(352.0, 704.0, 7.5, 7.5)

在 forward 中，

```
scores = bottom[0].data[:, self._num_anchors:, :, :]
bbox_deltas = bottom[1].data
im_info = bottom[2].data[0, :]

if DEBUG:
    print 'im_size: ({}, {})'.format(im_info[0], im_info[1])
    print 'scale: {}'.format(im_info[2])

# 1. Generate proposals from bbox deltas and shifted anchors
height, width = scores.shape[-2:]

if DEBUG:
    print 'score map size: {}'.format(scores.shape)

# Enumerate all shifts
shift_x = np.arange(0, width) * self._feat_stride
shift_y = np.arange(0, height) * self._feat_stride
shift_x, shift_y = np.meshgrid(shift_x, shift_y)
shifts = np.vstack((shift_x.ravel(), shift_y.ravel(),
                    shift_x.ravel(), shift_y.ravel())).transpose()
```

Bottom[0][1][2]分别指的是 Layer 10、Layer 9_2 和原图（224*224*3）的数据，而我的理解是 height 和 weight 分别为 14 和 14，下面 shift_x,shift_y 经过如下的过程：

Shift_x = [0,1,2,3,4,.....,width-1]*feat_stride,行向量

Shift_y = {[0],[1],[2],[3],[4],.....,[height-1]}*feat_stride，列向量

经过 meshgrid

$$\text{Shift\_x} = \begin{bmatrix} 0 & \cdots & \text{width}-1 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \text{width}-1 \end{bmatrix} * \text{feat\_stride}$$，共 height 行

$$\text{Shift\_y} = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \text{height}-1 & \cdots & \text{height}-1 \end{bmatrix} * \text{feat\_stride}$$，共 width 行

由测试文件可得：feat_stride=16（因为 14*16 刚好为 224，原图片大小，坐标变成原始图片能够适应的大小），因此：

$$\text{Shift\_x} = \begin{bmatrix} 0 & \cdots & 13 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 13 \end{bmatrix} * 16$$，共 13 行（感觉不对，但是又不知道咋写，因为 bottom 格式还没弄清楚）

$$\text{Shift\_y} = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 13 & \cdots & 13 \end{bmatrix} * 16$$，共 13 行

最后一行程序要经过 3 个变换，可得到下面的结果

$$\text{Shifts} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ w-1 & 0 & w-1 & 0 \\ 0 & 1 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ w-1 & 1 & w-1 & 1 \\ * & * & * & * \\ 0 & h-1 & 0 & h-1 \\ \vdots & \vdots & \vdots & \vdots \\ w-1 & h-1 & w-1 & h-1 \end{bmatrix} * 16$$

接下来就要和 anchor 有关了：

```
A = self._num_anchors
K = shifts.shape[0]
anchors = self._anchors.reshape((1, A, 4)) + \
          shifts.reshape((1, K, 4)).transpose((1, 0, 2))
anchors = anchors.reshape((K * A, 4))


bbox_deltas = bbox_deltas.transpose((0, 2, 3, 1)).reshape((-1, 4))

scores = scores.transpose((0, 2, 3, 1)).reshape((-1, 1))

# Convert anchors into proposals via bbox transformations
proposals = bbox_transform_inv(anchors, bbox_deltas)

# 2. clip predicted boxes to image
proposals = clip_boxes(proposals, im_info[:2])
```

A=9,K=width*height,经过了一系列的 reshape 和 transpose 得到 K*A，4 两个参数，一个表示 anchor 的个数，一个是参数，具体过程怎么做的，还没看明白，接下来是对 10 层和 9——2 层的两个结果进行 reshape，具体过程也没看明白，

然后经过两个函数，一个是得到预测的窗口,根据函数实现，得到公式

$$\mathrm{pred_x} = \mathrm{dx} \times \mathrm{width} + \mathrm{ctr_x}$$

$$\mathrm{pred_y} = \mathrm{dy} \times \mathrm{height} + \mathrm{ctr}_y$$

$$\mathrm{pred_w} = \mathrm{e}^{dw} \times \mathrm{width}$$
$$\mathrm{pred_h} = \mathrm{e}^{dh} \times \mathrm{height}$$

其中，dx，dy，dw，dh 为第 10 层的结果，ctrx，ctry，width，height 为 anchors 的参数，这个过程的意思就是将坐标扩展到原函数中，因为经过卷积和压缩，特征集尺度变小了。

最后再转化成四个坐标型（上面是中心加尺度型），另一个函数是保证上面的结果大于 0，且在图像中，没有溢出。

之后再经过一个_filter_boxes 函数去除尺度小于 min_size 的 boxes，剩下的位置保存在 keep 变量里面，然后将 Proposals 和配套的 scores 进行排序，将得分大于 0.7 的拿出来，例如：共 300 个。至此成功生成一定量的 proposals。最终的 top 的数据一部分是得分，一部分是坐标。

抱歉，这几层有多少个加，多少个减我已经数不清楚了