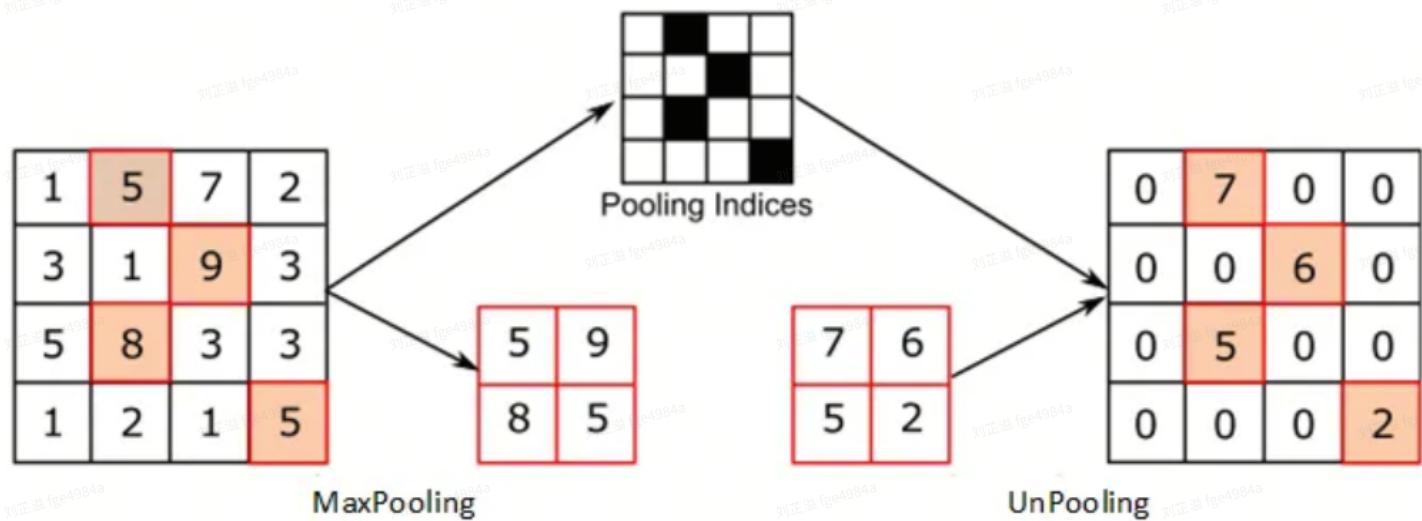


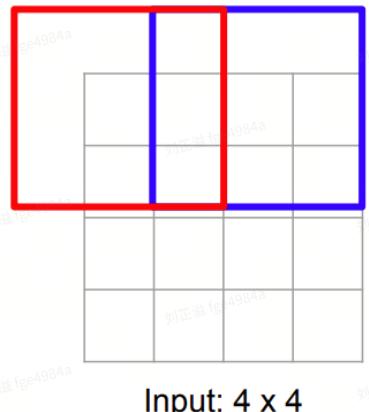
# Object Detection and Image Segmentation

## Unpooling

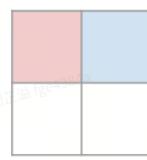


## Downsampling

**Recall:** Normal  $3 \times 3$  convolution, stride 2 pad 1



Dot product  
between filter  
and input



Input:  $4 \times 4$

Output:  $2 \times 2$

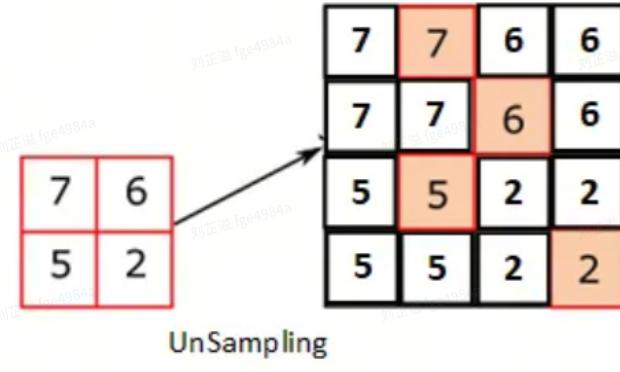
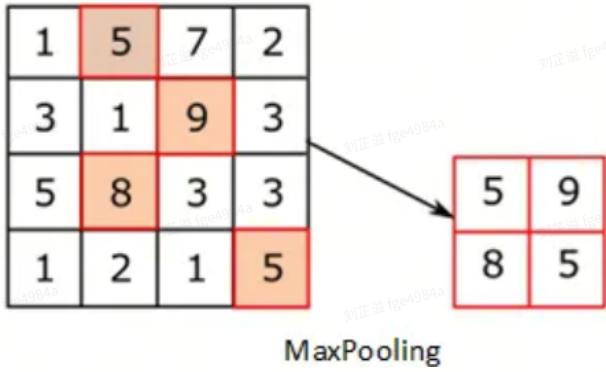
Filter moves 2 pixels in  
the input for every one  
pixel in the output

Stride gives ratio between  
movement in input and  
output

We can interpret strided  
convolution as “learnable  
downsampling”.

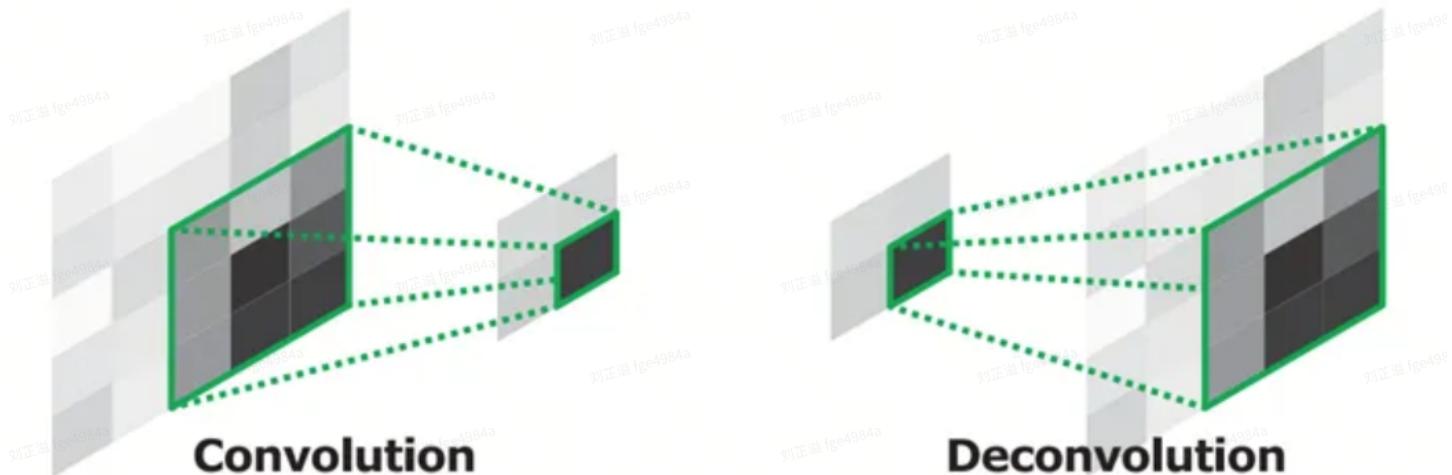
## Upsampling

- 将所有位置都赋值成特征图中的值



## 反卷积

- 逆卷积相对于卷积在神经网络结构的正向和反向传播中做相反的运算。参数需要学习
- 逆卷积(Deconvolution)比较容易引起误会，转置卷积(Transposed Convolution)是一个更为合适的叫法。



## Two-Stage Object Detection

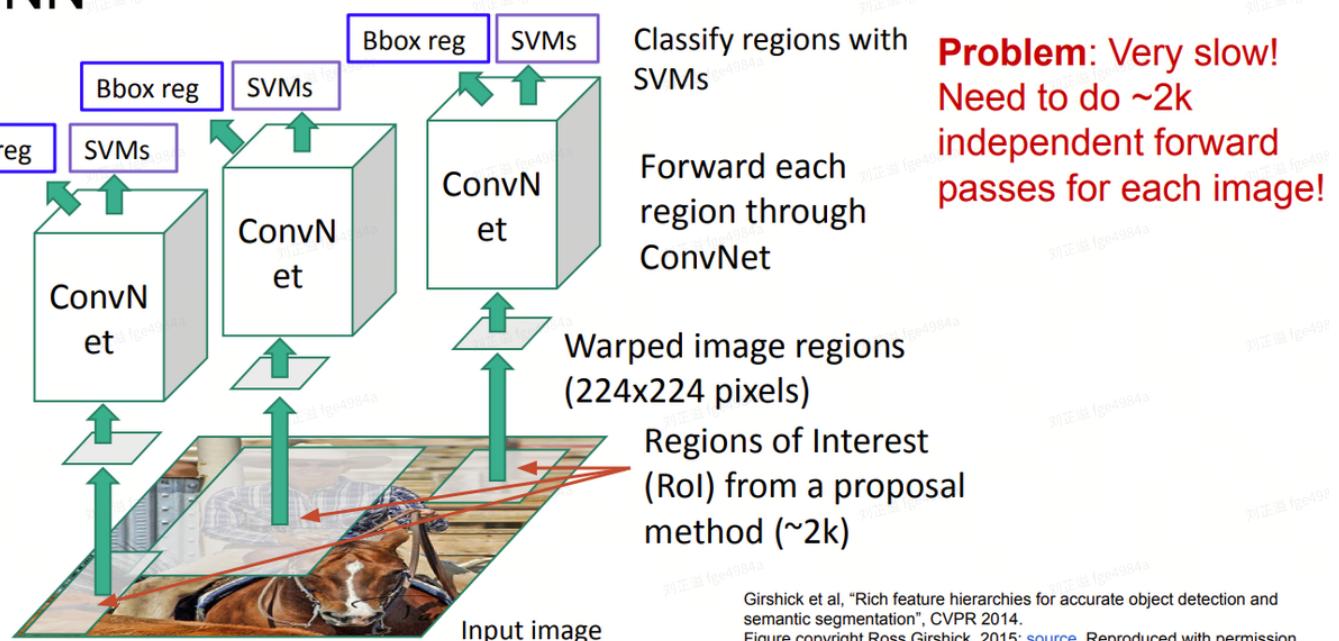
- ## First stage: Run once per image
- Backbone network
  - Region proposal network

- ## Second stage: Run once per region
- Crop features: RoI pool / align
  - Predict object class
  - Prediction bbox offset

### R-CNN

- 先画框，对每个画完框的图片进行卷积
- 对卷积后的图片进行SVM和对框的分类

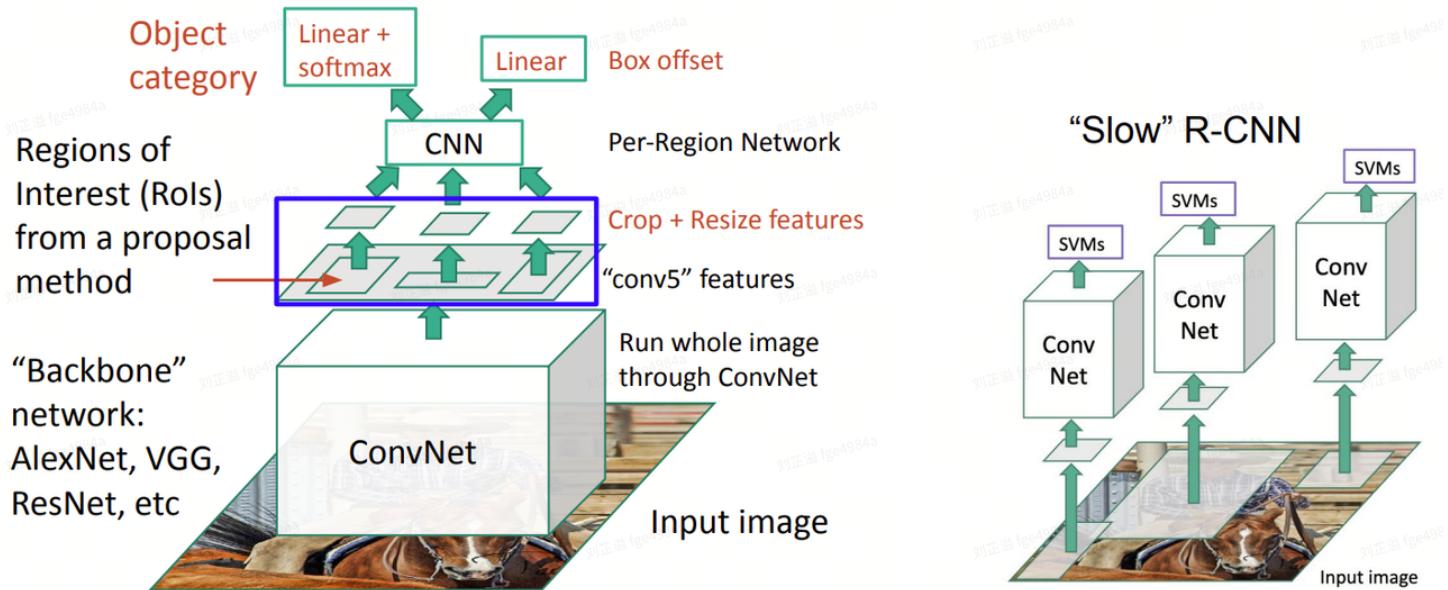
### R-CNN



### Fast R-CNN

- 使用RoI Pool进行图像框的筛选
- RoI Pool对图像框进行裁剪

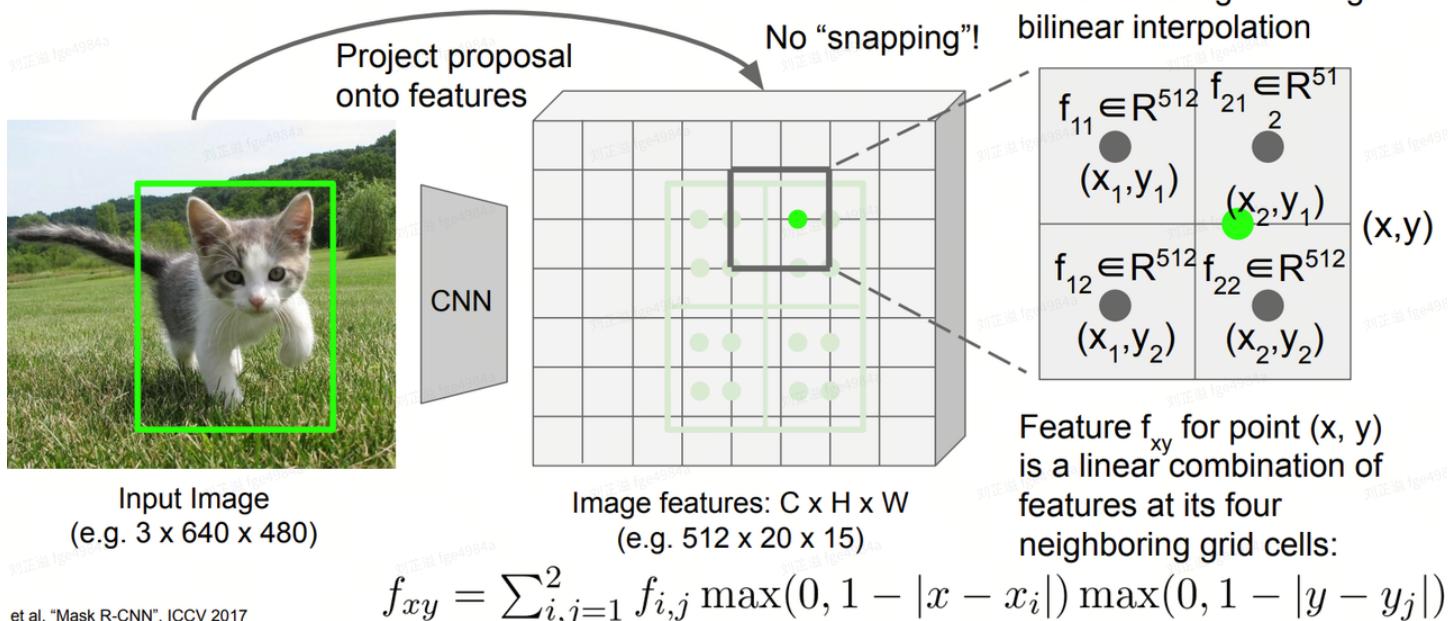
# Fast R-CNN



## RoI Align

- 点(x,y)的值由周围的4个点线性组合来表示

## Cropping Features: RoI Align



## Faster R-CNN

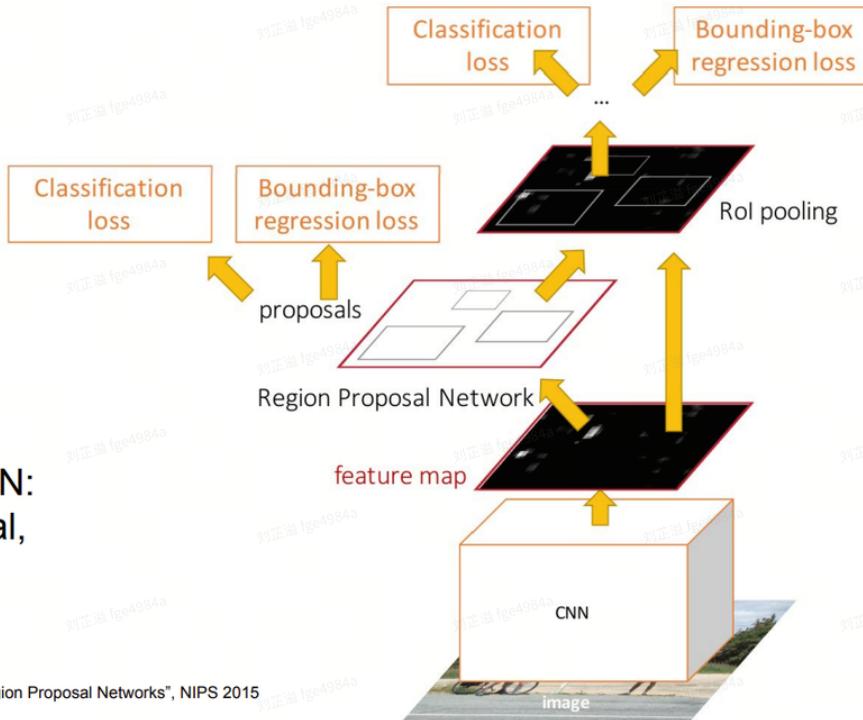
- 使用Region Proposal Network; 预测框中的特征

# Faster R-CNN:

Make CNN do proposals!

Insert Region Proposal Network (RPN) to predict proposals from features

Otherwise same as Fast R-CNN:  
Crop features for each proposal,  
classify each one



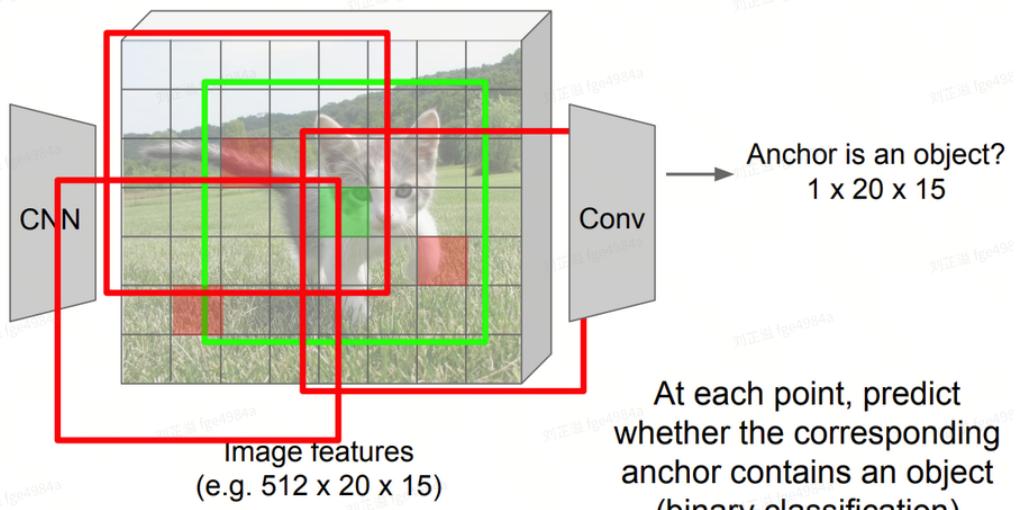
## RPN

- 选定K种不同大小的框，对框的客观性分数进行排序，取前300作为我们的建议
- 对每个点进行框中是否有目标的二元分类
- 对正框还需要进行真实框与选定框的修正

## Region Proposal Network



Input Image  
(e.g.  $3 \times 640 \times 480$ )



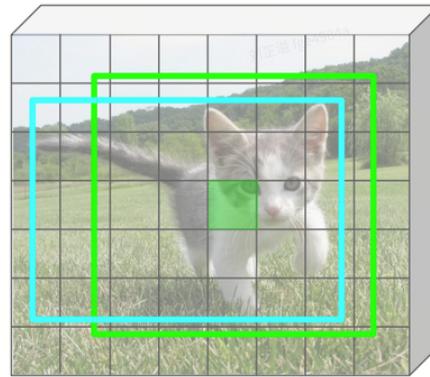
At each point, predict whether the corresponding anchor contains an object (binary classification)

Imagine an **anchor box** of fixed size at each point in the feature map

# Region Proposal Network



Input Image  
(e.g.  $3 \times 640 \times 480$ )



Imagine an **anchor box**  
of fixed size at each  
point in the feature map

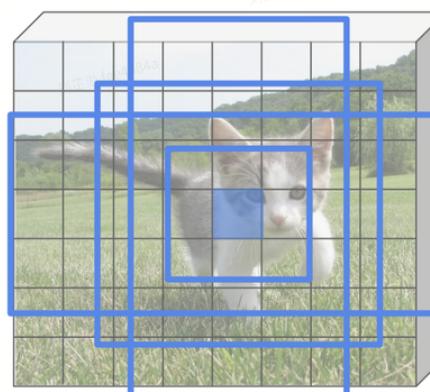
→ Anchor is an object?  
 $1 \times 20 \times 15$   
→ Box corrections  
 $4 \times 20 \times 15$

For positive boxes, also predict  
a corrections from the anchor to  
the ground-truth box (regress 4  
numbers per pixel)

# Region Proposal Network



Input Image  
(e.g.  $3 \times 640 \times 480$ )



In practice use  $K$  different  
anchor boxes of different  
size / scale at each point

→ Anchor is an object?  
 $K \times 20 \times 15$   
→ Box transforms  
 $4K \times 20 \times 15$

Sort the  $K \times 20 \times 15$  boxes by  
their “objectness” score,  
take top  $\sim 300$  as our  
proposals

## 4 losses

- RPN是否有目标
- RPN回归盒子坐标
- 最后的分类分数
- 最后的盒子坐标

# Jointly train with 4 losses:

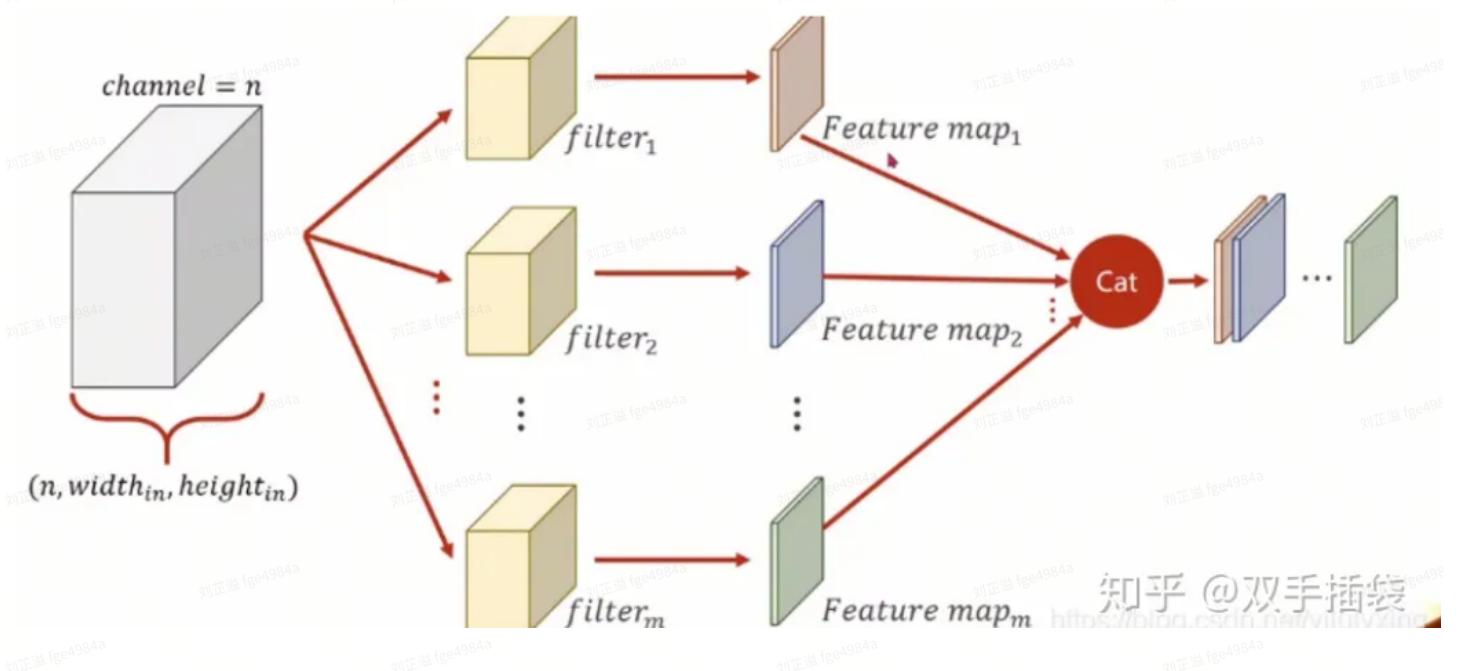
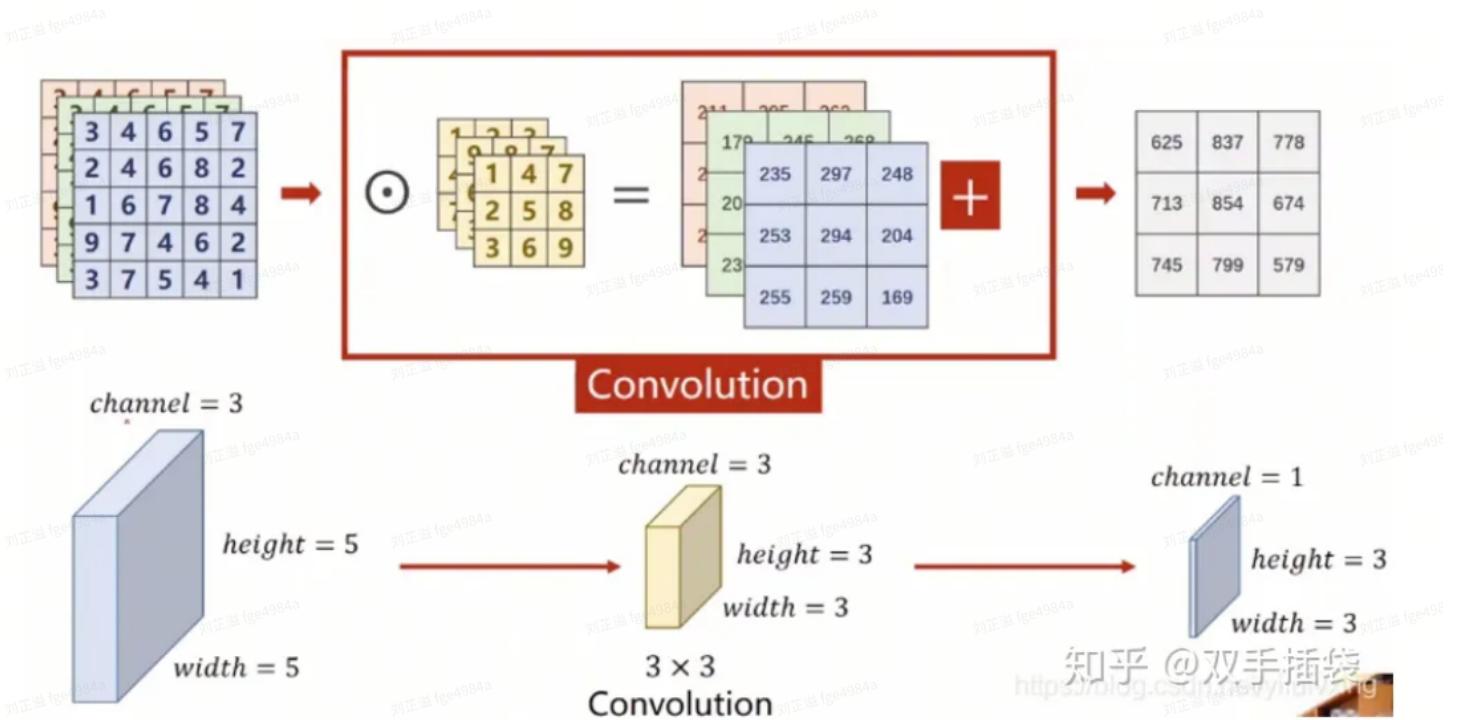
1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates

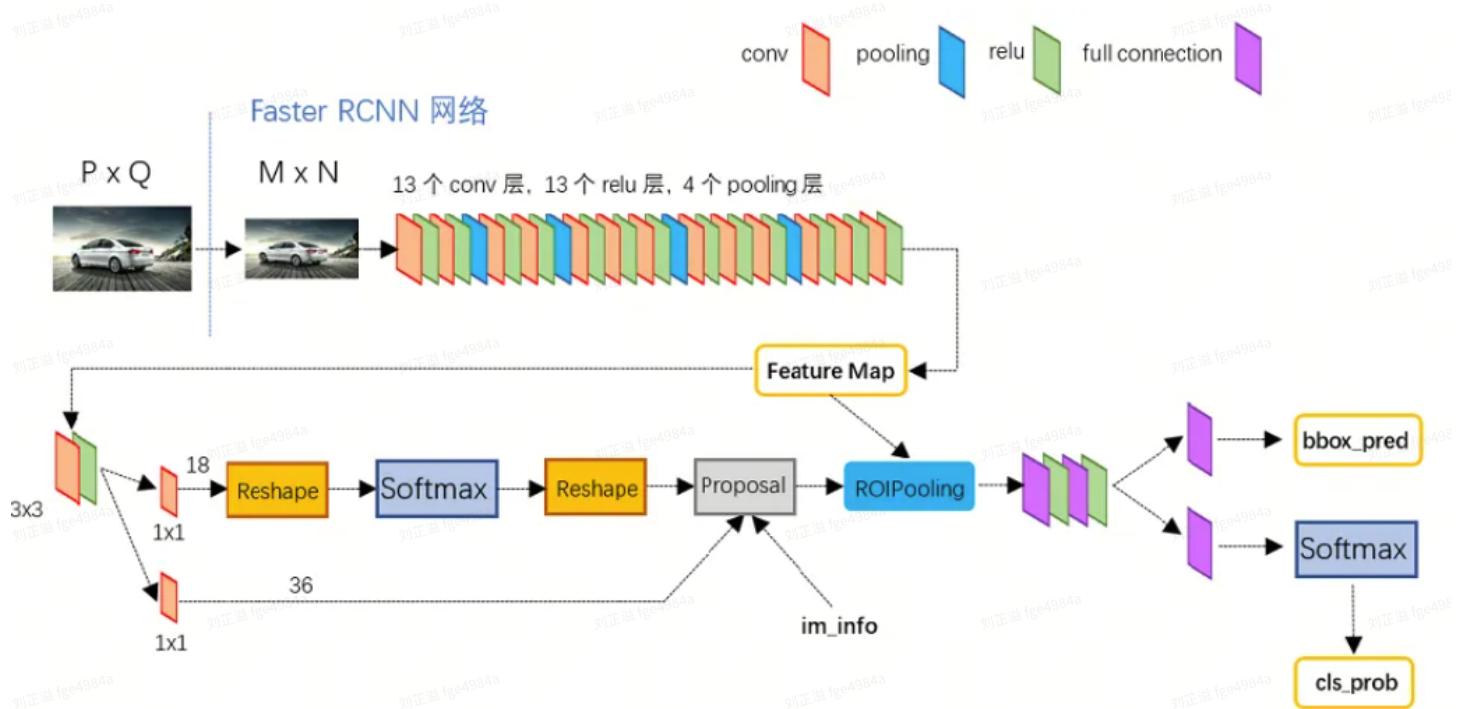
## Details

- 忽略重叠框的非极大值抑制算法（NMS）
- 网络主体
  1. Conv layers。作为一种CNN网络目标检测方法，Faster RCNN首先使用一组基础的conv+relu+pooling层提取image的feature maps。该feature maps被共享用于后续RPN层和全连接层。
  2. Region Proposal Networks。RPN网络用于生成region proposals。该层通过softmax判断anchors属于positive或者negative，再利用bounding box regression修正anchors获得精确的proposals。
  3. Roi Pooling。该层收集输入的feature maps和proposals，综合这些信息后提取proposal feature maps，送入后续全连接层判定目标类别。
  4. Classification。利用proposal feature maps计算proposal的类别，同时再次bounding box regression获得检测框最终的精确位置。

## Conv layers

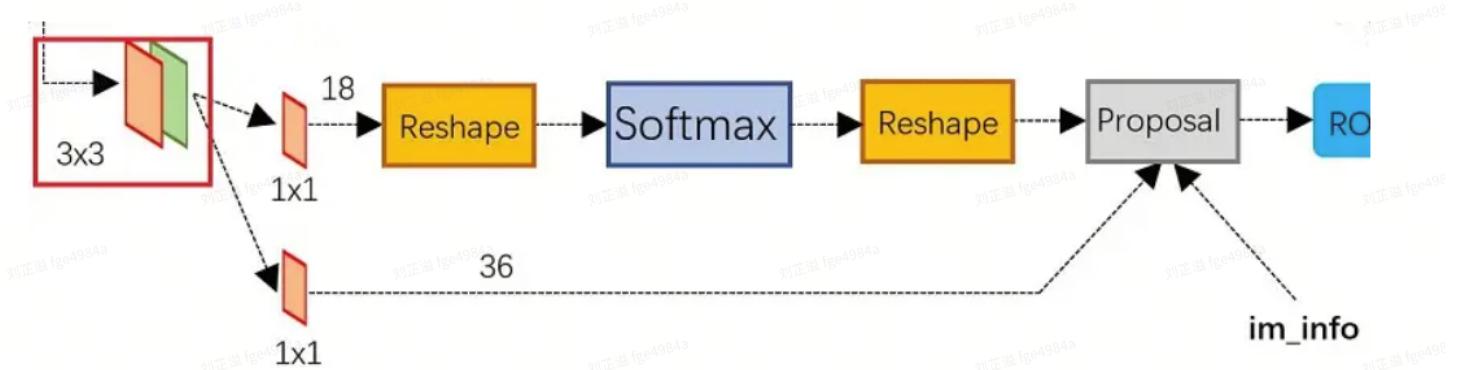
1. 所有的conv层都是：kernel\_size=3, pad=1, stride=1
  - 该设置不改变输入和输出矩阵的大小
2. 所有的pooling层都是：kernel\_size=2, pad=0, stride=2
  - 这样每个经过pooling层的MxN矩阵，都会变为(M/2)x(N/2)大小。综上所述，在整个Conv layers中，conv和relu层不改变输入输出大小，只有pooling层使输出长宽都变为输入的1/2。
  - 那么，一个MxN大小的矩阵经过Conv layers固定变为(M/16)x(N/16)！这样Conv layers生成的feature map中都可以和原图对应起来。
- 多通道卷积输出通道数量取决于卷积核的数量





## RPN

- 经典的检测方法生成检测框都非常耗时，如OpenCV adaboost使用滑动窗口+图像金字塔生成检测框；或如R-CNN使用SS(Selective Search)方法生成检测框。而Faster RCNN则抛弃了传统的滑动窗口和SS方法，直接使用RPN生成检测框，这也是Faster R-CNN的巨大优势，能极大提升检测框的生成速度。
- RPN网络实际分为2条线：
  - 上面一条通过softmax分类anchors获得positive和negative分类
  - 下面一条用于计算对于anchors的bounding box regression偏移量，以获得精确的proposal。
  - 而最后的Proposal层则负责综合positive anchors和对应bounding box regression偏移量获取 proposals，同时剔除太小和超出边界的proposals。其实整个网络到了Proposal Layer这里，就完成了相当于目标定位的功能。



## anchors

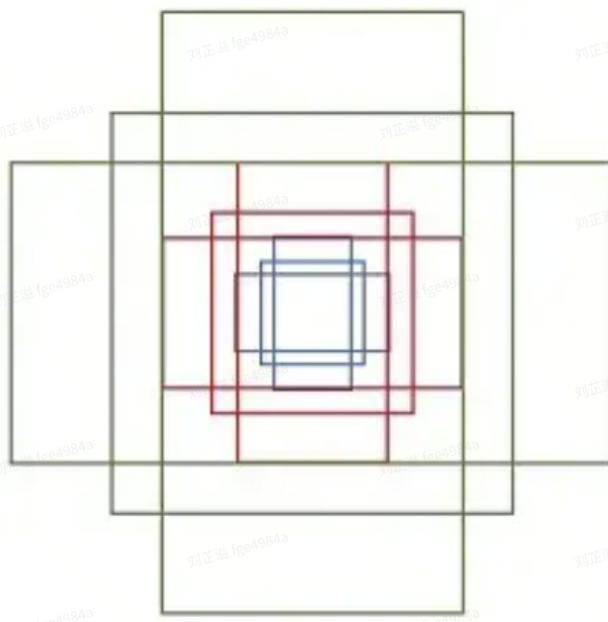
- 把任意大小的输入图像reshape成 $800 \times 600$ （即图2中的 $M=800$ ,  $N=600$ ）。再回头来看anchors的大小，anchors中长宽1:2中最大为 $352 \times 704$ ，长宽2:1中最大 $736 \times 384$ ，基本是cover了 $800 \times 600$ 的各个尺度和形状。

```

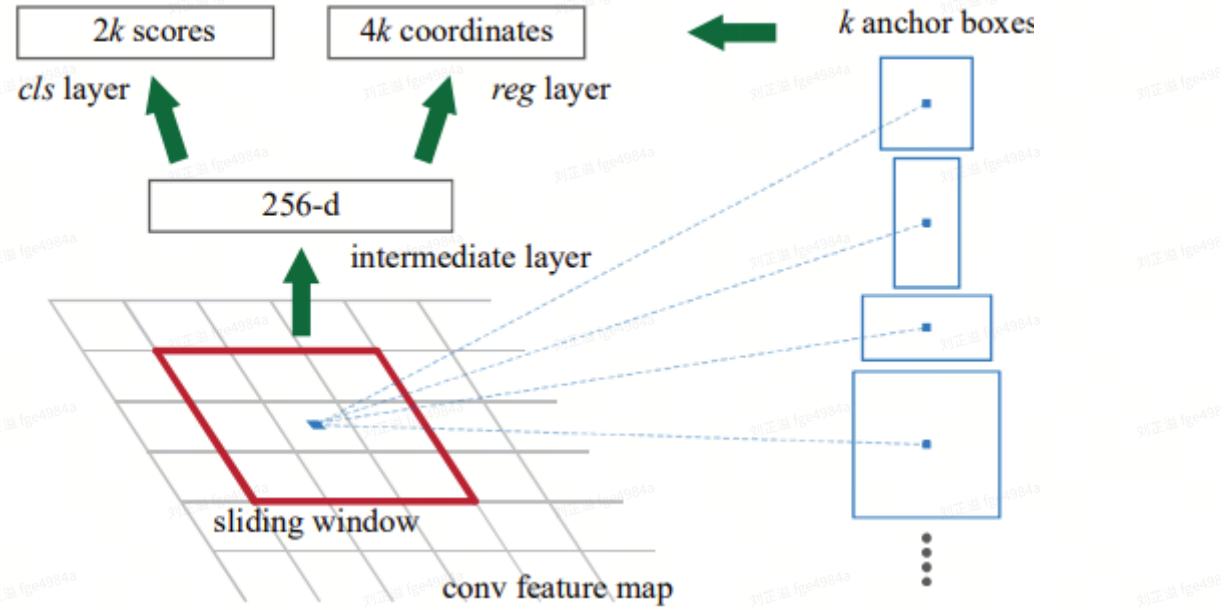
[[ -84.  -40.   99.   55.]
 [-176. -88.  191.  103.]
 [-360. -184.  375.  199.]
 [-56.  -56.   71.   71.]
 [-120. -120.  135.  135.]
 [-248. -248.  263.  263.]
 [-36.  -80.   51.   95.]
 [-80. -168.   95.  183.]
 [-168. -344.  183.  359.]]

```

其中每行的4个值  $(x_1, y_1, x_2, y_2)$  表矩形左上和右下角点坐标。9个矩形共有3种形状，长宽比为大约为  $\text{width:height} \in \{1:1, 1:2, 2:1\}$  三种，如图6。实际上通过anchors就引入了检测中常用到的多尺度方法。



- 遍历Conv layers计算获得的feature maps，为每一个点都配备这9种anchors作为初始的检测框。这样做获得检测框很不准确，不用担心，后面还有2次bounding box regression可以修正检测框位置
- feature map中每个点上有k个anchor（默认k=9），而每个anhcor要分positive和negative，所以每个点由256d feature转化为 $\text{cls}=2 \cdot k \text{ scores}$ ；而每个anchor都有 $(x, y, w, h)$ 对应4个偏移量，所以 $\text{reg}=4 \cdot k \text{ coordinates}$
- 全部anchors拿去训练太多了，训练程序会在合适的anchors中随机选取128个postive anchors+128个negative anchors进行训练**



- 经过该卷积的输出图像为WxHx18大小（注意第二章开头提到的卷积计算方式）。这也就刚好对应了feature maps每一个点都有9个anchors，同时每个anchors又有可能是positive和negative，所有这些信息都保存WxHx(9\*2)大小的矩阵

### Bounding box regression

即经过该卷积输出图像为WxHx36，在caffe blob存储为[1, 4x9, H, W]，这里相当于feature maps每个点都有9个anchors，每个anchors又都有4个用于回归的变换量。

$$\text{Loss} = \sum_i^N |t_*^i - W_*^T \cdot \phi(A^i)| \quad (7)$$

函数优化目标为：

$$\hat{W}_* = \operatorname{argmin}_{W_*} \sum_i^n |t_*^i - W_*^T \cdot \phi(A^i)| + \lambda ||W_*|| \quad (8)$$

为了方便描述，这里以L1损失为例介绍，而真实情况中一般使用smooth-L1损失。

需要说明，只有在GT与需要回归框位置比较接近时，才可近似认为上述线性变换成立。

说完原理，对应于Faster RCNN原文，positive anchor与ground truth之间的平移量 ( $t_x, t_y$ ) 与尺度因子 ( $t_w, t_h$ ) 如下：

$$t_x = (x - x_a)/w_a \quad t_y = (y - y_a)/h_a \quad (9)$$

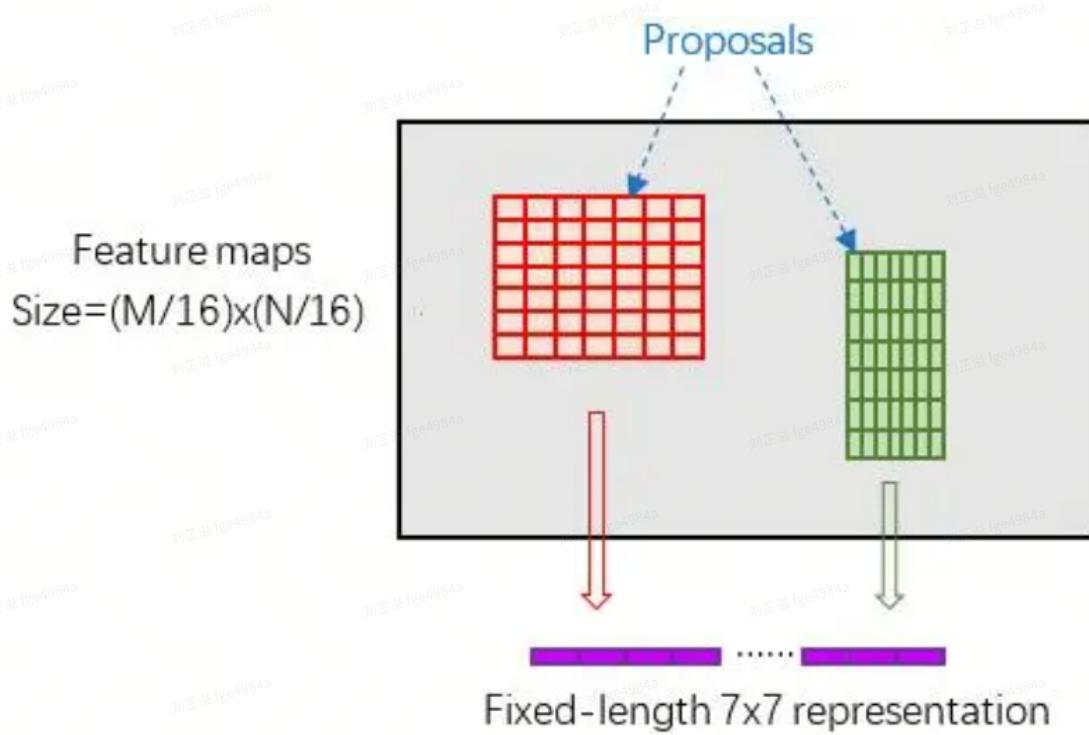
$$t_w = \log(w/w_a) \quad t_h = \log(h/h_a) \quad (10)$$

## Proposal Layer

- 生成anchors，对所有的anchors做bbox regression回归（这里的anchors生成和训练时完全一致）
- 按照输入的positive softmax scores由大到小排序anchors，提取前pre\_nms\_topN(e.g. 6000)个anchors，即提取修正位置后的positive anchors
- 限定超出图像边界的positive anchors为图像边界，防止后续roi pooling时proposal超出图像边界
- 剔除尺寸非常小的positive anchors
- 对剩余的positive anchors进行NMS (nonmaximum suppression)
- Proposal Layer有3个输入：positive和negative anchors分类器结果rpn\_cls\_prob\_reshape，对应的bbox reg的(e.g. 300)结果以及im\_info作为proposal输出
  - 对于一副任意大小PxQ图像，传入Faster RCNN前首先reshape到固定MxN，im\_info=[M, N, scale\_factor]则保存了此次缩放的所有信息。然后经过Conv Layers，经过4次pooling变为WxH=(M/16)x(N/16)大小，其中feature\_stride=16则保存了该信息，用于计算anchor偏移量。

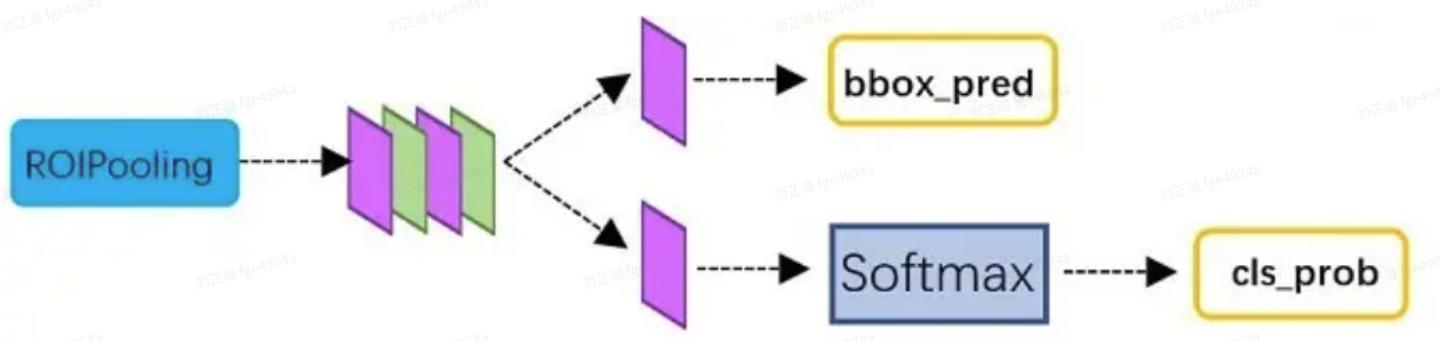
## Roi Pooling

- 由于proposal是对应MxN尺度的，所以首先使用spatial\_scale参数将其映射回(M/16)x(N/16)大小的feature map尺度；
- 再将每个proposal对应的feature map区域水平分为 pool\_w\*pool\_h 的网格；
- 对网格的每一份都进行max pooling处理。
- 这样处理后，即使大小不同的proposal输出结果都是 pool\_w\*pool\_h 固定大小，实现了固定长度输出。



## Classification

- Classification部分利用已经获得的proposal feature maps，通过full connect层与softmax计算每个proposal具体属于那个类别（如人，车，电视等），输出cls\_prob概率向量；
- 同时再次利用bounding box regression获得每个proposal的位置偏移量bbox\_pred，用于回归更加精确的目标检测框。



## Training

与检测网络类似的是，依然使用Conv Layers提取feature maps。整个网络使用的Loss如下：

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \quad (12)$$

上述公式中  $i$  表示anchors index， $p_i$  表示positive softmax probability， $p_i^*$  代表对应的GT predict概率（即当第*i*个anchor与GT间IoU>0.7，认为是该anchor是positive， $p_i^* = 1$ ；反之IoU<0.3时，认为是该anchor是negative， $p_i^* = 0$ ；至于那些0.3<IoU<0.7的anchor则不参与训练）； $t$  代表predict bounding box， $t^*$  代表对应positive anchor对应的GT box。可以看到，整个Loss分为2部分：

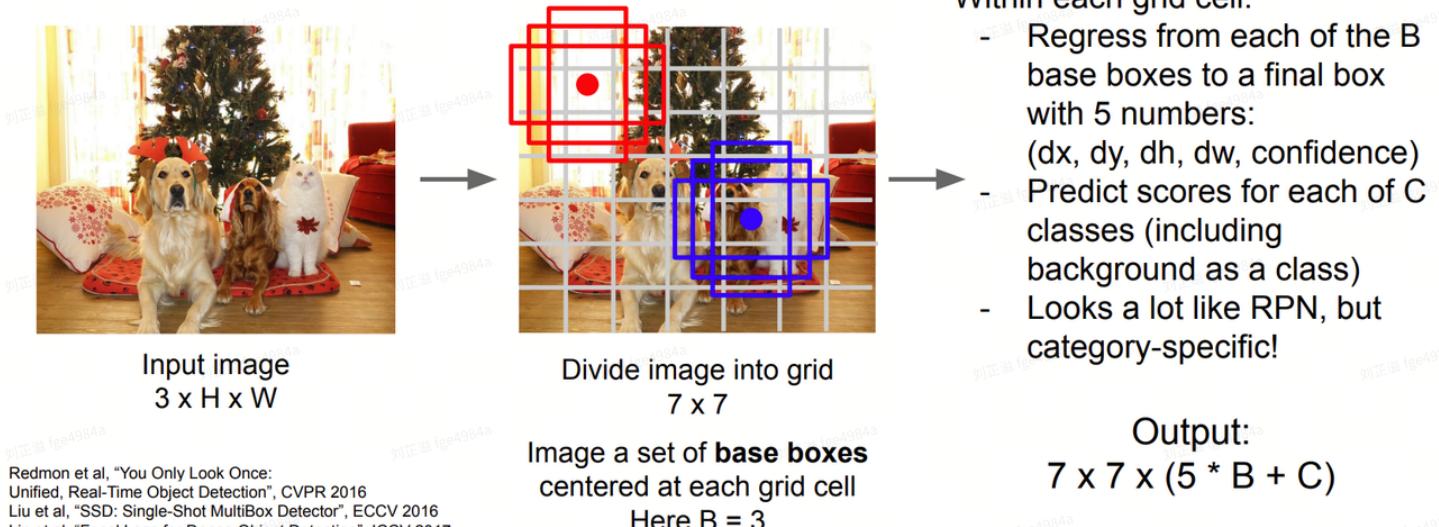
1. cls loss，即rpn\_cls\_loss层计算的softmax loss，用于分类anchors为positive与negative的网络训练
2. reg loss，即rpn\_loss\_bbox层计算的smooth L1 loss，用于bounding box regression网络训练。注意在该loss中乘了  $p_i^*$ ，相当于只关心positive anchors的回归（其实在回归中也完全没必要去关心negative）。

由于在实际过程中， $N_{cls}$  和  $N_{reg}$  差距过大，用参数 $\lambda$ 平衡二者（如 $N_{cls} = 256$ ,  $N_{reg} = 2400$ 时设置  $\lambda = \frac{N_{reg}}{N_{cls}} \approx 10$ ），使总的网络Loss计算过程中能够均匀考虑2种Loss。这里比较重要是

1. 训练RPN网络
2. 通过训练好的RPN网络收集proposals
3. 训练Faster RCNN网络

# Single-Stage Object Detectors

## Single-Stage Object Detectors: YOLO / SSD / RetinaNet



### YOLO

- YOLO仅利用卷积层，使其成为一个全卷积网络（FCN）。在YOLO v3论文中，作者提出了一个名为Darknet-53的更深的特征提取器架构。正如其名称所示，它包含53个卷积层，每个卷积层后面跟随批量归一化层和Leaky ReLU激活函数。
- 没有使用任何形式的池化，而使用带有步长2的卷积层来降采样特征图。这有助于**防止池化经常归因于低级特征的丢失**。