简介

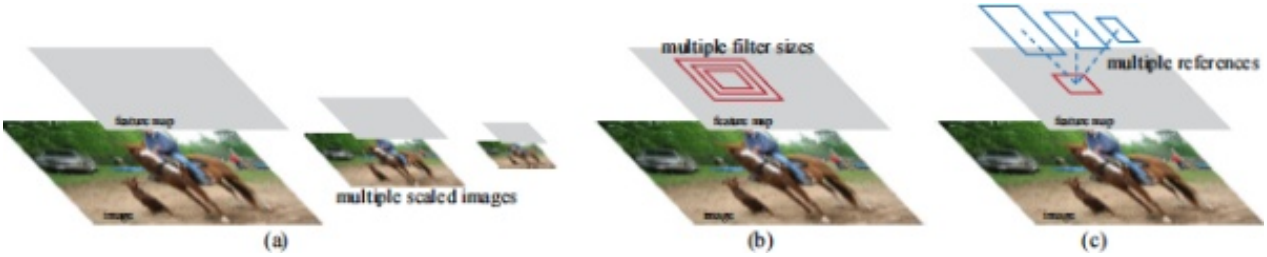
Faster R-CNN 论文地址《Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks》，该论文发表于 2015年，同时论文也给出了实验代码caffe版本 py-faster-rcnn，TensorFlow 版本的代码地址 Faster-RCNN_TF。论文中引入了 RPN（Region Proposal Network） 网络，它是一种全卷积神经网络 FCN（Fully Convolutional Network），同时训练使用端对端的方式来生成高质量的候选区域。使用卷积神经网络来计算 proposals，以此来加速计算。

这篇文章阅读的难度超出了预期，迷失在很多细节之上，一度产生了怀疑为什么有这么多资料还是没有弄明白。知道你去接触事情的根源，即阅读源代码，所有的事情才算明了。这篇论文的阅读收获不少，写文章时把自己讲述的基础提一下是多么重要。在另一篇文章《Faster R-CNN 论文阅读记录（二）：细节》里会详细的描述Faster R-CNN 的训练细节，对于理解其他解读文章会更有帮助。

具体详细的解读可以参考下面文章，三篇文章综合起来看会对 Faster R-CNN 有较全面的了解

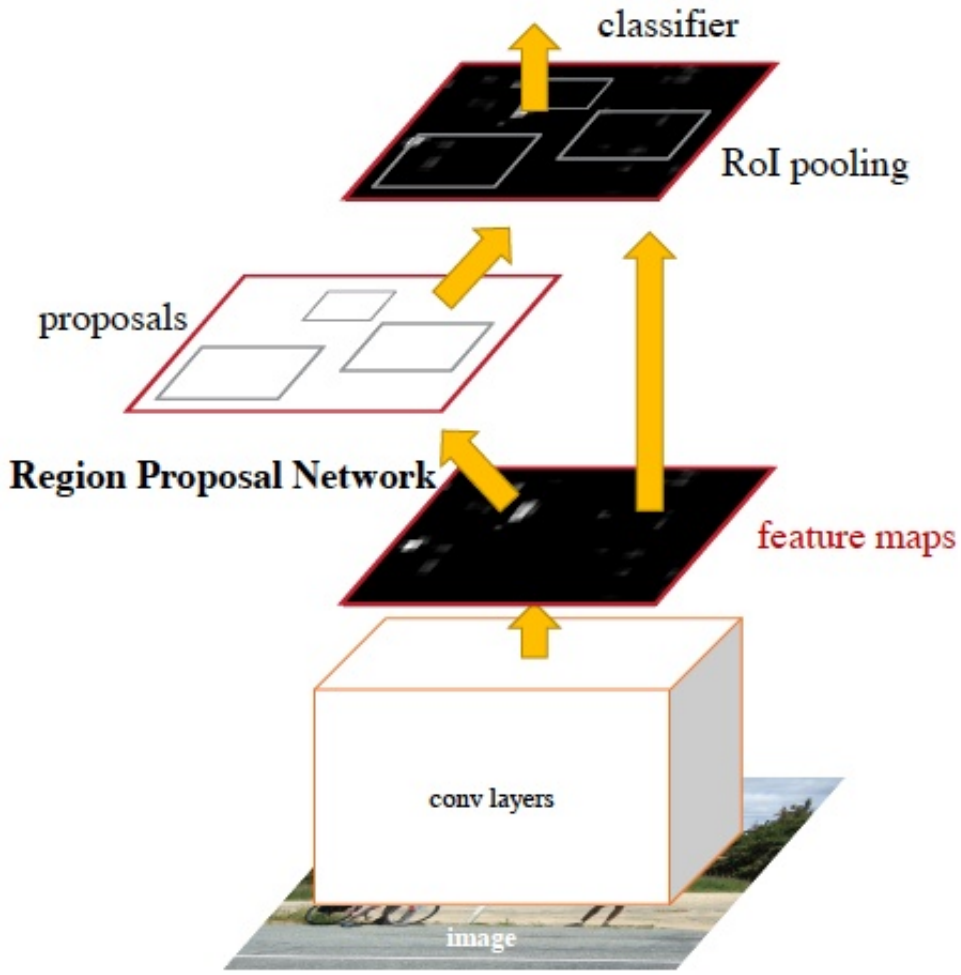
- cnblogs - Faster R-CNN
- 知乎 - 一文读懂Faster RCNN
- 个站 - Object Detection and Classification using R-CNNs
- CSDN - faster-rcnn 原理解析
- cnblogs - Faster R-CNN论文详解

图像金字塔、过滤器（filter）金字塔、锚点（anchor boxes）比较



(a) Pyramids of images and feature maps are built, and the classifier is run at all scales.
(b) Pyramids of filters with multiple scales/sizes are run on the feature map.
(c) We use pyramids of reference boxes in the regression functions.
anchor boxes 避免了枚举多尺寸和长宽比的图片和过滤器（filter）。该模型在使用单尺寸图片训练与测试时都获得了较好的表现。

结构



Faster R-CNN 的主要过程如下：

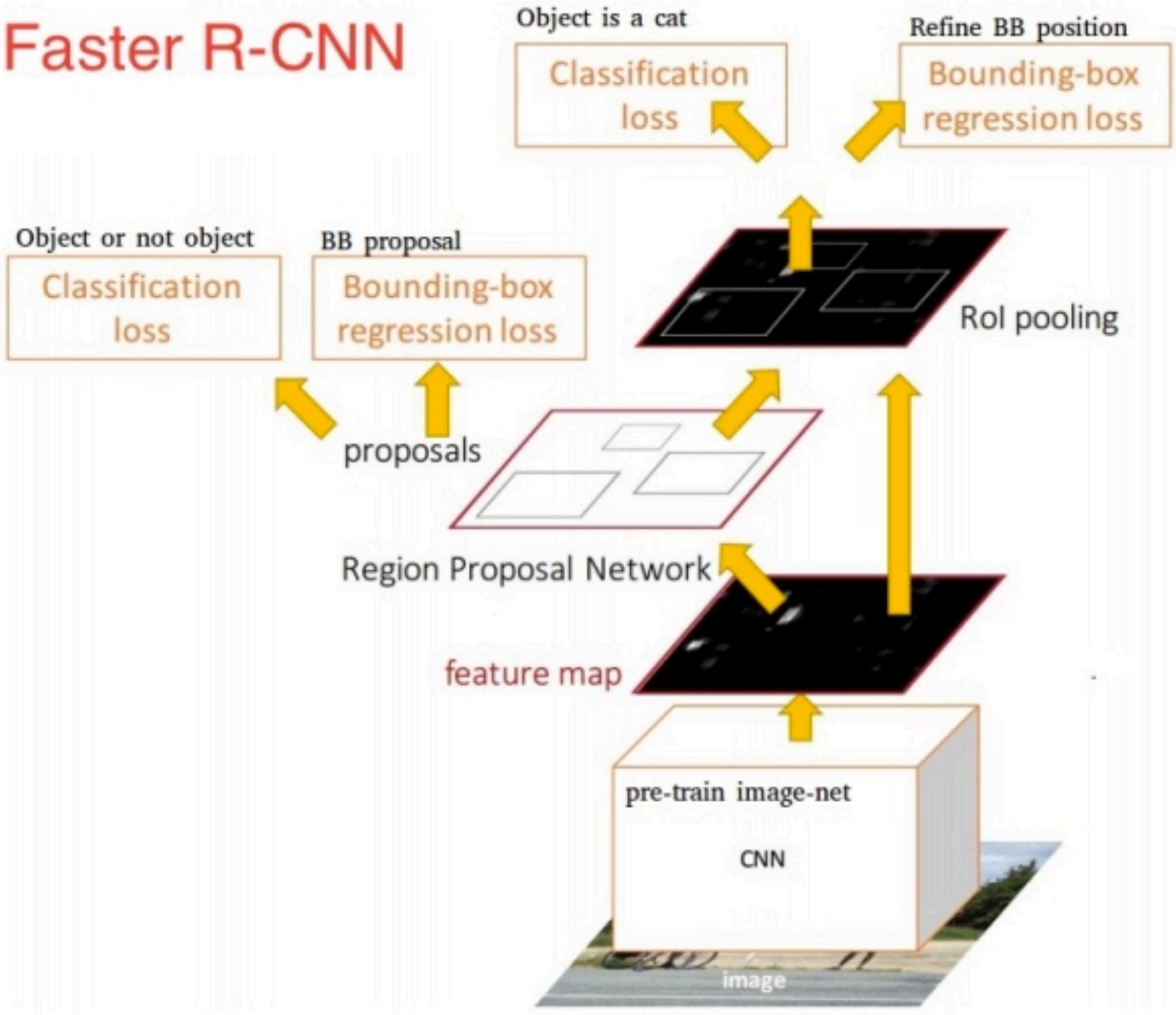
1. conv layers。作为一种CNN网络目标检测方法，Faster RCNN首先使用一组基础的conv+relu+pooling层提取image的feature maps。该feature maps被共享用于后续RPN层和全连接层。

2. Region Proposal Networks。RPN网络用于生成region proposals。该层通过softmax判断anchors属于foreground或者background，再利用bounding box regression修正anchors获得精确的proposals。

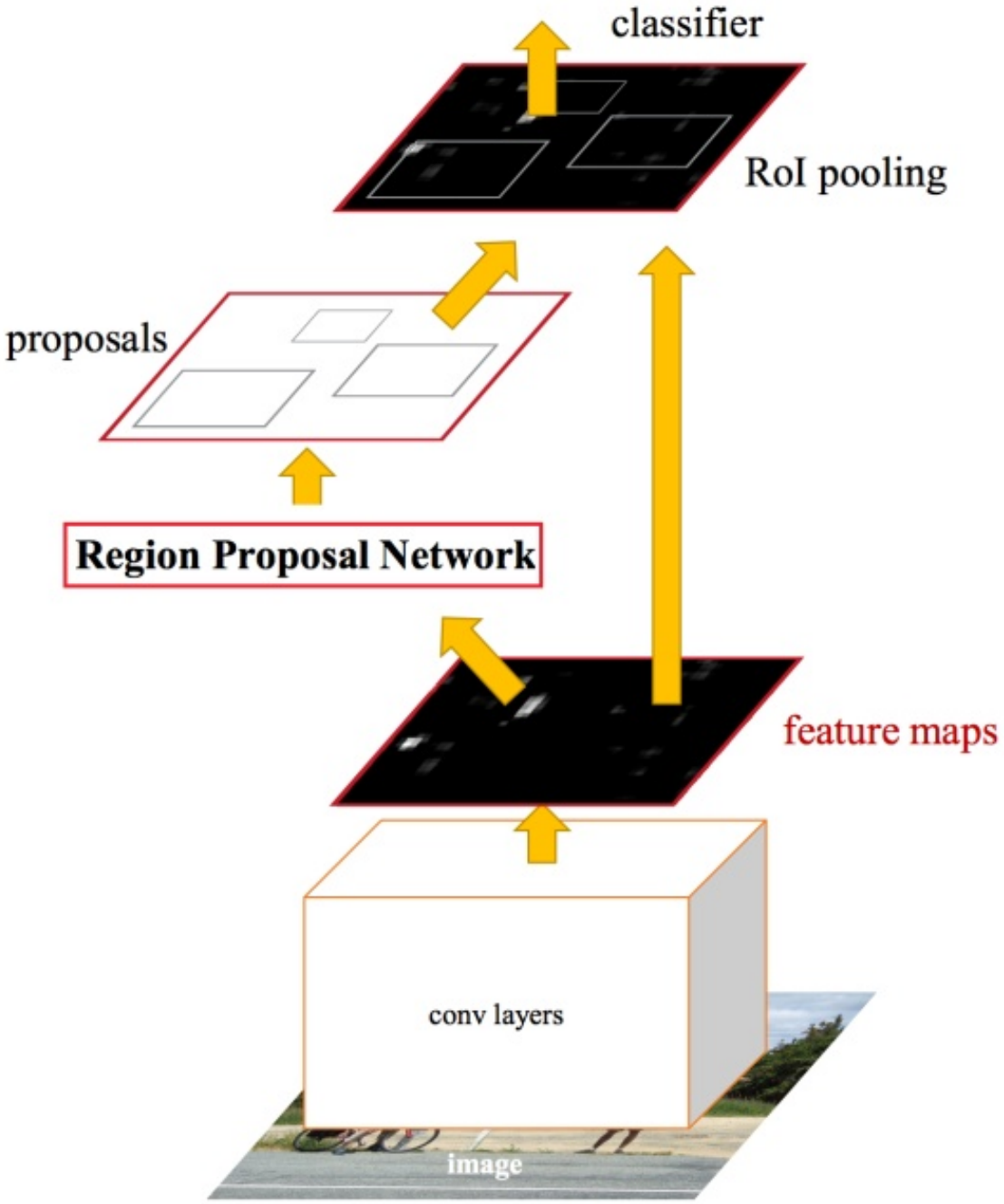
3. Roi Pooling。该层收集输入的feature maps和proposals，综合这些信息后提取proposal feature maps，送入后续全连接层判定目标类别。

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。

对于结构的一下理解，在网上可以搜到下面一张结构图：



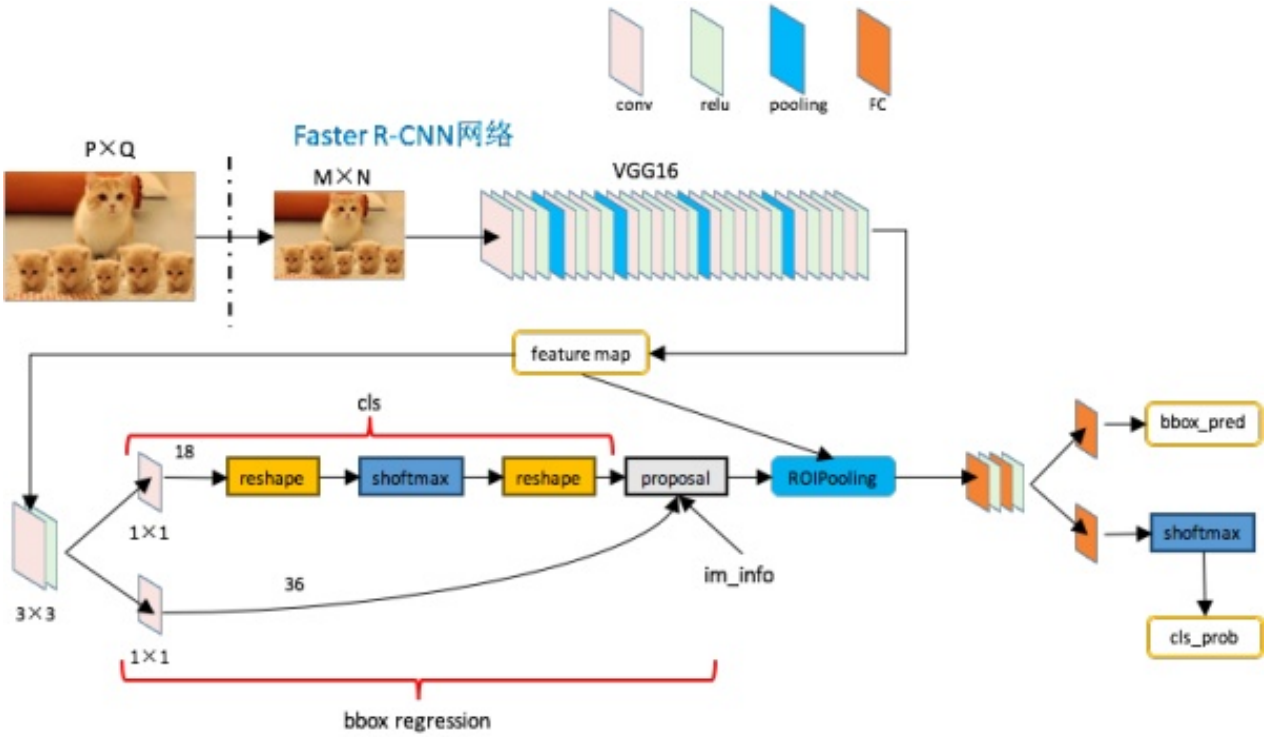
我觉着这张图应该是有些问题的，上面的“object or not object” 和 “BB proposal” 应该是在 Region Proposal Network 中完成的，而不是又在输出的 proposals 进行这些操作，RPN 的输出结果就是 proposals。下面这张图更应该能反应真实的情况：



RPN 输入任意尺寸的图片，返回一组带有objectness score 的矩形候选区。objectness score 用来衡量物体类别（object classes）与背景（background）之间的关系。此过程可以通过 FCN 来模拟。

网络的目标是为了让 RPN 与 Fast R-CNN 可以共享计算，这里就假设两个网络共享了通用的卷积层。在论文的实验中，作者研究了 ZFNet 和 VGG-16，前者有5个可共享的卷积层，后者含有13个可共享的卷积层。VGG-16有13个可共享的卷积层，VGG-16一共有16个含参卷积层，去掉最后的3个 FC 还剩 13 个卷积层。

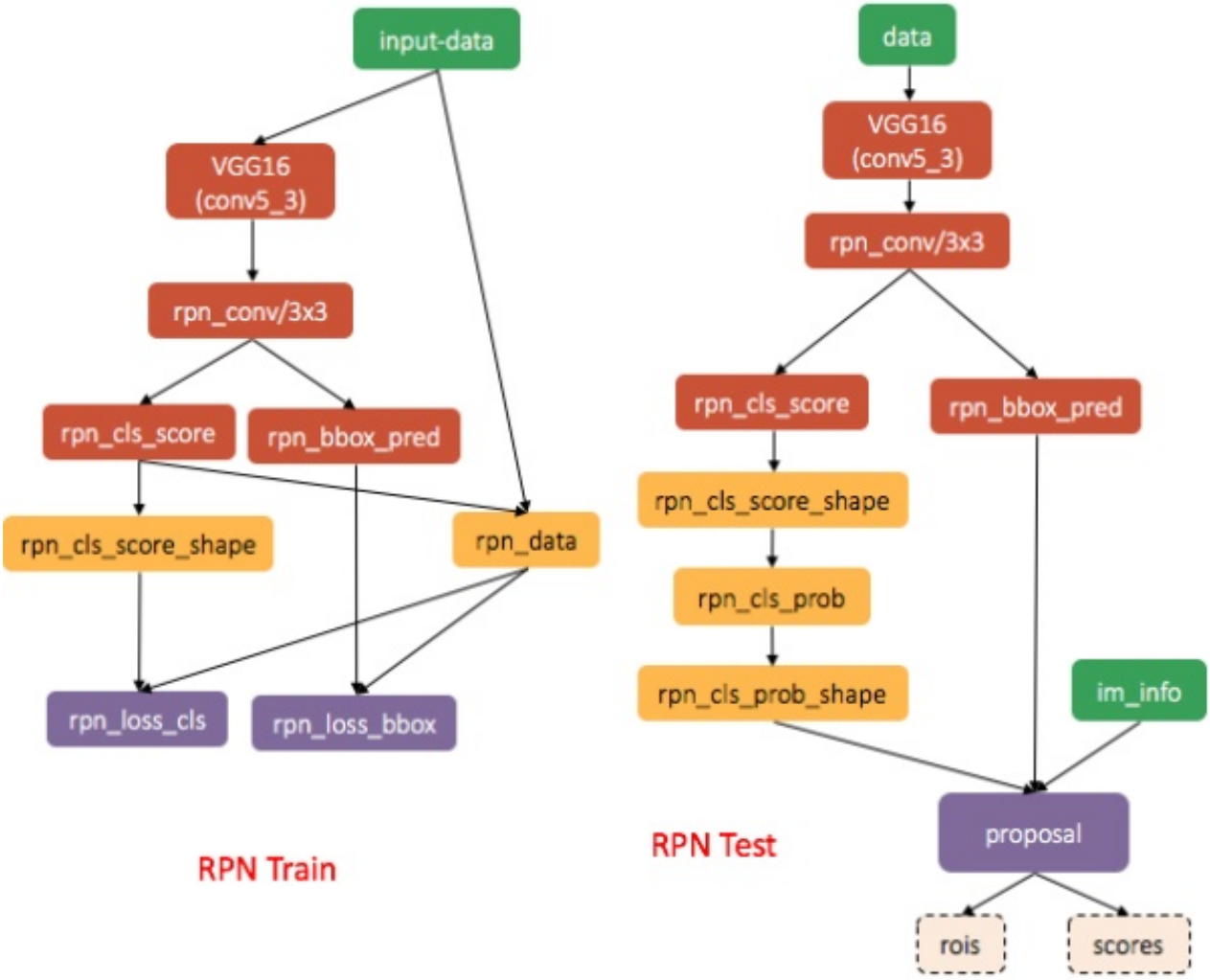
Faster R-CNN 更细节的结构图：



RPN 结构

主要用于提取候选区域，速度比 SS（selective search） 和 EB（edgebox） 要快很多，结构主要使用了 FCN，rpn_cls_score 和 rpn_bbox_pred 为 1x1 卷积。在 caffe 中的结构如下（以下内容来自 cnblogs - Faster R-CNN）：

```
rpn_conv/3x3 num output:512
// (4个尺度的候选框, 64 128 256 512, 3中宽高比, 共12个候选框)
// 论文中使用的是 3 个尺度，这里多了个 64
rpn_cls_score num output:24 (前景概率+背景概率)x12
rpn_bbox_pred num output:48 (四个坐标) x 12
rpn_loss_cls: SoftmaxLoss
rpn_loss_bbox: SmoothL1
```



左图是 RPN 训练时的结构，这时输入的数据是有标签的，包含了 data、im_info、gt_boxes数据，rpn_data 负责产生 RPN 训练所需要的数据。右图为 RPN 测试时使用的结构，这时的数据是没有标签的，RPN 的输出是 rois 和 scores。网上大部分给出的结构都是 end2end 方式训练的结构图，也有的给出的是非 end2end 的图，在阅读的时候需要注意到这一点。

上面的 objectness cls 分类器只判断框中有没有物体，而不判断是哪一类物体，这个判断需要交给 Fast R-CNN 来做。

[参考]

- CSDN - 深度学习: RPN (区域候选网络)
- 个站 - Notes on Faster RCNN

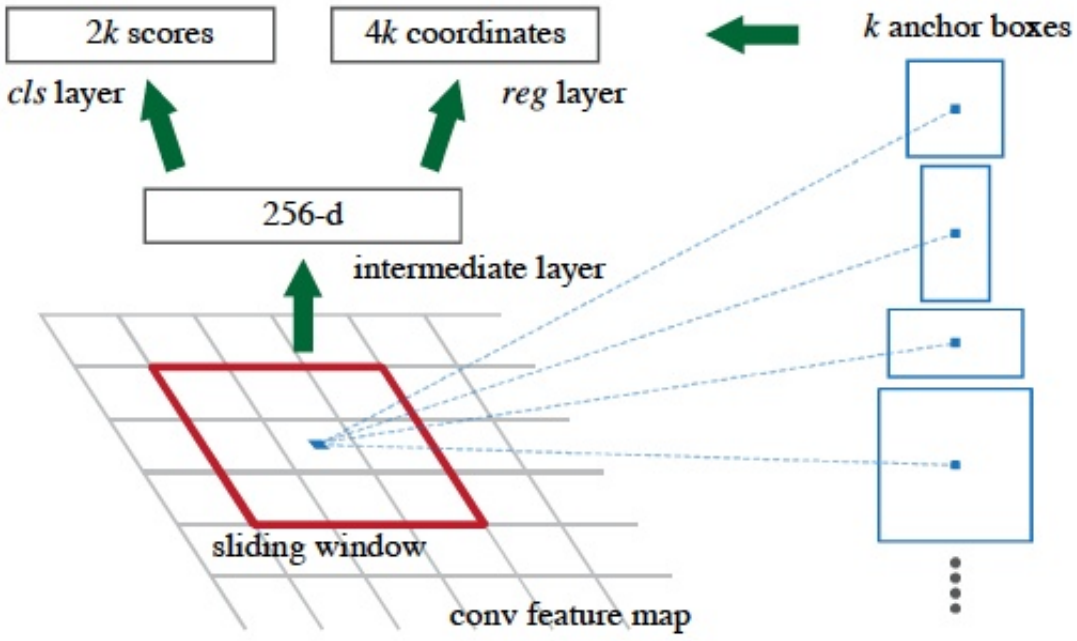
Anchors

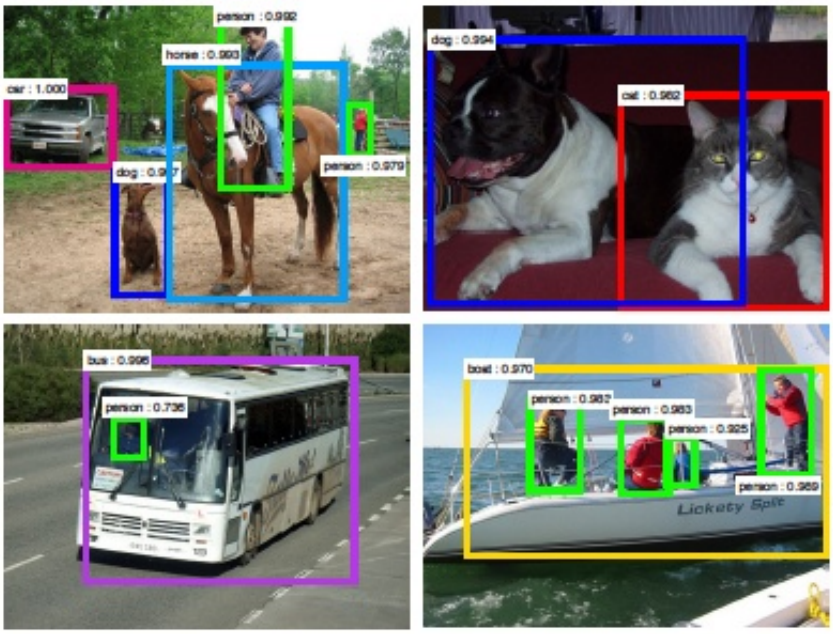
在每个滑动窗口位置，同时预测多个候选区（region proposals），每个位置最大可能的候选区域数记为 k 。因此对于 reg 层需要 4k 个输出来编码 k 个盒子的坐标（四个点），cls 层则需要 2k 个输出分数，来评估每个区域是目标和不是目标的概率。论文中 cls 层的实现是使用 two-class softmax。

k 个建议是通过相对于 k 个推断 box 来参数化，其中的推断 box 就被称为 anchor。anchor 是滑动窗口的中心，他与尺度（scale）和长宽比（aspect ratio）相关联。 默认情况下使用 3 个 scale 和 3 个长宽比，因此会产生 9 个 anchor 在每个滑动位置。对于一个 W×H 大小的卷积特征映射，会产生 W×H×K 个 anchor。之所以得到 W×H×K 个 anchor ，是因为 3×3 的卷积网络在 W×H 上滑动是进行了

padding，且值为 1 ，stride 为 1 。那么就会产生 W×H 此滑动，每次滑动有 K 个 anchor，那么最终就产生了 W×H×K 个 anchor 。

RPN 中 Anchor 产生的示意图如下：





anchor 的优点: 它只依赖与单个 scale 的 images 和 feature map 不过却能解决 multiple scales and sizes 的问题。

训练

Faster R-CNN 的训练分为四步，不断循环四步训练网络，过程如下图：

参考《知乎 - 一文读懂Faster RCNN》重新绘制

训练涉及到的细节比价多，这里不详细的讲解，在[《Faster R-CNN 论文阅读记录（二）：细节》]一文中会有更多的交待。

实验总结

使用 RPN 生成的 proposals，彼此之间有很多的重叠。为了减少冗余，就需要依赖 cls 得到的分值使用 NMS（non-maximum suppression） 进行削减。 最终留下 2000 个 proposals。使用 NMS 之后，使用 top-N 排序候选区域进行检测。

不同的 proposal 产生方法对比图

train-time region proposals		test-time region proposals		
method	# boxes	method	# proposals	mAP (%)
SS	2000	SS	2000	58.7
EB	2000	EB	2000	58.6
RPN+ZF, shared	2000	RPN+ZF, shared	300	**59.9**
ablation experiments follow below				
RPN+ZF, unshared	2000	RPN+ZF, unshared	300	58.7
SS	2000	RPN+ZF	100	55.1
SS	2000	RPN+ZF	300	56.8
SS	2000	RPN+ZF	1000	56.3
SS	2000	RPN+ZF (no NMS)	6000	55.2
SS	2000	RPN+ZF (no *cls*)	100	44.6
SS	2000	RPN+ZF (no *cls*)	300	51.4
SS	2000	RPN+ZF (no *cls*)	1000	55.8
SS	2000	RPN+ZF (no *reg*)	300	52.1
SS	2000	RPN+ZF (no *reg*)	1000	51.3
SS	2000	RPN+VGG	300	59.2

可以看到使用 RPN 的 Fast R-CNN 的 mAP 达到了 59.9%，而且只用了 300 个 proposals，且速度还比 SS（ Selective Search） 和 EB （ EdgeBoxes）的方法快很多。如下图：

model	system	conv	proposal	region-wise	total	rate
VGG	SS + Fast R-CNN	146	1510	174	1830	0.5 fps
VGG	RPN + Fast R-CNN	141	**10**	47	**198**	**5 fps**
ZF	RPN + Fast R-CNN	31	**3**	25	**59**	**17 fps**

在 ablation 测试中，RPN + ZF 100 与 RPN + ZF（no NMS）有相似的 mAP，可以看出 NMS 对检测 mAP 并没有损害，可能减少了 false alarms 的数量。

在 cls上的比较中，可以看到 N= 1000时，mAP（55.8%） 并没有改变多少，但当 N= 100 时，mAP（44.6%） 就减少了很多。这说明 cls 分数考虑了最高排序建议的精确度（ This shows that the cls scores account for the accuracy of the highest ranked proposals）。

在 reg 的比较中，可以看到 N=1000或者300，mAP 并没有明显的变化，说明 anchor boxes 对检测的精确度并没有充足的保证，虽然有多个尺度和比例。说明 reg 是产生高质量 proposals 的关键。

使用 VGG16 结构

使用 RPN 与 VGG16结合的结果如下图：

method	# proposals	data	mAP (%)
SS	2000	07	66.9†
SS	2000	07+12	70.0
RPN+VGG, unshared	300	07	68.5
RPN+VGG, shared	300	07	69.9
RPN+VGG, shared	300	07+12	73.2
RPN+VGG, shared	300	COCO+07+12	78.8

超参的敏感性

论文中研究了 anchor 不同的设置，如下图：

settings	anchor scales	aspect ratios	mAP (%)
1 scale, 1 ratio	128^2	1:1	65.8
	256^2	1:1	66.7
1 scale, 3 ratios	128^2	{2:1, 1:1, 1:2}	68.8
	256^2	{2:1, 1:1, 1:2}	67.9
3 scales, 1 ratio	$\{128^2, 256^2, 512^2\}$	1:1	69.8
3 scales, 3 ratios	$\{128^2, 256^2, 512^2\}$	{2:1, 1:1, 1:2}	69.9

可以看到 3 scales 1 ratio 和 3scales 3 ratios 的检测精度非常接近，这就说明 scale 和 ratio 并不是不可分开的维度对于检测精度来说。

损失函数中 λ 的影响

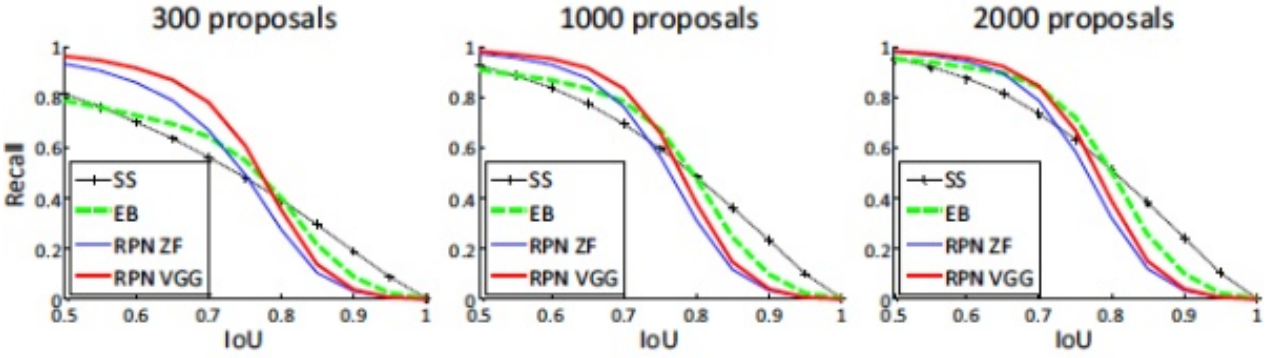
损失公式为：

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

λ	0.1	1	10	100
mAP (%)	67.2	68.9	69.9	69.1

可以看到为 10 的时候 mAP 的分值最高。也表明结果对 λ 在大范围内变化并不敏感。

Recall 与 IoU



可以看到 RPN 的 proposal 从 2000 降到 300 行为是非常的优雅的，几乎没有怎么改变。这也表明了 cls 在其中起到了很大的作用。SS 和 EB 随着 proposal 的减少，下降速度非常的快。

One-Stage Detection vs. Two-Stage Proposal + Detection

OverFeat 是 one-stage, class-specific 检测的 pipeline，而 Faster R-CNN 是 two-stage cascade consisting of class-agnostic proposals and class-specific detections。

	proposals		detector	mAP (%)
Two-Stage	RPN + ZF, unshared	300	Fast R-CNN + ZF, 1 scale	58.7
One-Stage	dense, 3 scales, 3 aspect ratios	20000	Fast R-CNN + ZF, 1 scale	53.8
One-Stage	dense, 3 scales, 3 aspect ratios	20000	Fast R-CNN + ZF, 5 scales	53.9

问题

RPN 中为什么选择 3×3 的滑动窗口

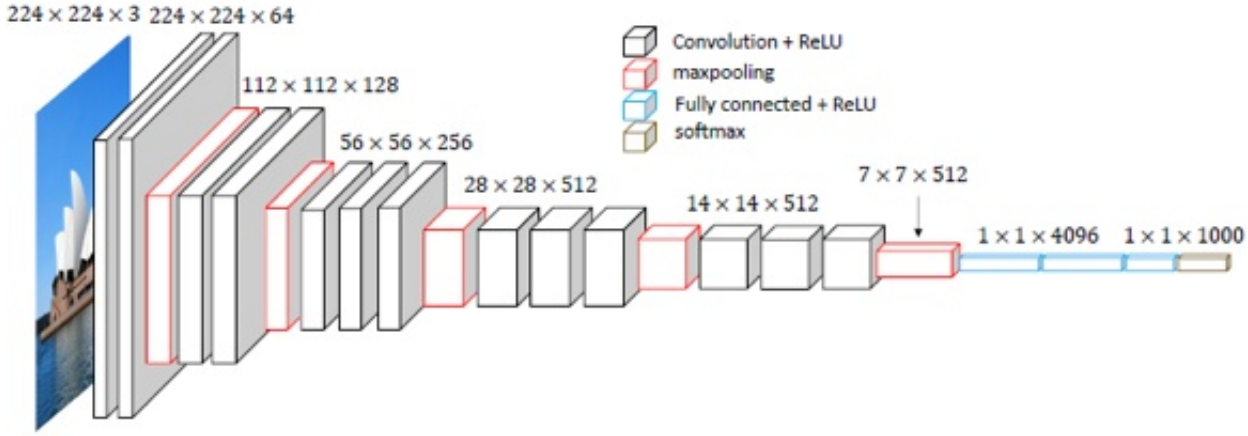
RPN 选择了3×3 的滑动窗口来做卷积，那么为什么是 3×3 而不是其他的尺寸呢？这是因为：

> 1.卷积操作本身就是一个卷积核(滑动窗口)在特征图(图像)的滑动操作。
> 2.这里滑动窗口的目的是为了取每个窗口的特征：
> 对于VGG16，最后一个卷积层(conv5_3)的feature map个数为512，使用3x3的卷积核可以每张feature map上获取一个1x1的特征，总共是512的一个特征向量，这个向量也就是当前滑窗对应的特征。
> 3.为什么使用3x3的滑窗而不是2x2或者其他，个人理解：因为3x3在原图像的感受野是228x228，对于尺度为128，256，512的anchor设计来说，对于128x128的region proposal，228x228是个很不错的选择（包含了上下文信息），256尺度的跟228差不多， 512x512的只利用了中心的228x228的特征(虽然不是很好，但也凑合)，所以选择3x3的滑窗也算是一个技巧，目的是让这个滑动窗口的感受野跟region proposal的尽可能接近，这样去分类和做窗口回归才会更准。

其实上面也解释了为什么要将最小边缩放到 600，因为在论文中设定的面积最大的正方形的 anchor 的最长边恰好是 512 ，为了确保 anchor 能覆盖大部分图片，那么就需要让图片的大小刚好和最大 anchor 的边长接近，因此选择 600 是最佳的。

接收视野为什么是 228 像素?

在论文的 3.1 章节提到，在最终的共享卷积层输出的特征映射上，使用小的 n×n 滑动网络来生成区域建议。每个滑动窗口被映射到低维的特征，其中 VGG16 是 512 维，这里的 512 维应该指的是VGG16 最终输出的 512 个 filter。如下图：



当使用的 n=3 的滑动网络时，VGG16 在原始的大图上的接收视野为 228像素，那这个尺寸是如何算出来的呢？这里可以参考 CNN：接受视野（Receptive Field）这篇文章的计算公式

$$r_i = s_i \cdot (r_{i+1} - 1) + k_i$$

其中 r_i 表示第 i 层输入的一个区域， s_i 表示第 i 层的步长， k_i 表示第 i 层卷积核的大小（filter size）。此计算不需要考虑 padding size。

可以知道最顶层的输入 $r_{i+1} = 3$ ，那么根据 VGG16 的结构及其上面的公式，就可以一步步的推导出最终结果。通过推导可以发现，每经过一个卷积层视野就增加 2 个像素，每经过一个池化层视野扩大 2 倍。上面的结果都是在卷积层的 stride为1，池化层的 stride 为 2的基础上，那么推到过程如下：

```
3 --> 3×conv3_512 = 9 --> maxpool = 18 --> 3×conv3_512 = 24 --> maxpool = 48 -
-> 3×conv3_256 = 54 --> maxpool = 108 --> 2×conv3_128 = 112 --> maxpool = 224
--> 2×conv3_64 = 228
```

[参考]

- github - Why does it need a reshape layer in RPN's cls layer
- caffecn - py-faster rcnn中rpn的3x3的滑框用卷积层来定义的是为什么？

实现细节中 stride 步长的问题

看了网上资料有说 16 是一个超参，可以进行任意的选择，就像 10 与 375 像素。

```
The last line says: "accuracy can be improved by using smaller strides" so
that means 16 is arbitrarily chosen (a hyperparameter). The other part is
obvious if stride is 16 for 600 pixels then stride is ~10 for 375 pixels.
```

在 Faster R-CNN 的 GitHub 上也有类似的问题：

```
Changing stride length decreased object detection accuracy #294
```

但通过 fast r-cnn 论文的描述，和《leanote - 详解 ROI Align 的基本原理和实现细节》此文的描述，我更倾向于 16 是一个累积步长。以figure1 VGG16 为例，卷积层并没有改变输入的尺寸，只有经过池化层时才会让输入的尺寸减半，而 Faster R-CNN 恰好要经过 4 个池化层，因此累积的结果就是 2×2×2×2 = 16。

对于 ZFNet，只有步长为 2 的前四层会让尺寸减半，因此总的 stride 也是 16：

网络层	卷积核尺寸	步长	填充	输出维数
conv1	7×7	2	3	96
pool1	3×3	2	1	-
conv2	5×5	2	2	256
pool2	3×3	2	1	-
conv3	3×3	1	1	384
conv4	3×3	1	1	384
conv5	3×3	1	1	256
fc6	-	-	-	4096
fc7	-	-	-	4096

在 PASCAL 上为缩放的图片最小边为 375，通过扩大 1.6 倍达到 600，那么相应的将 原始的 16 缩小 1.6 倍就是 10 像素。

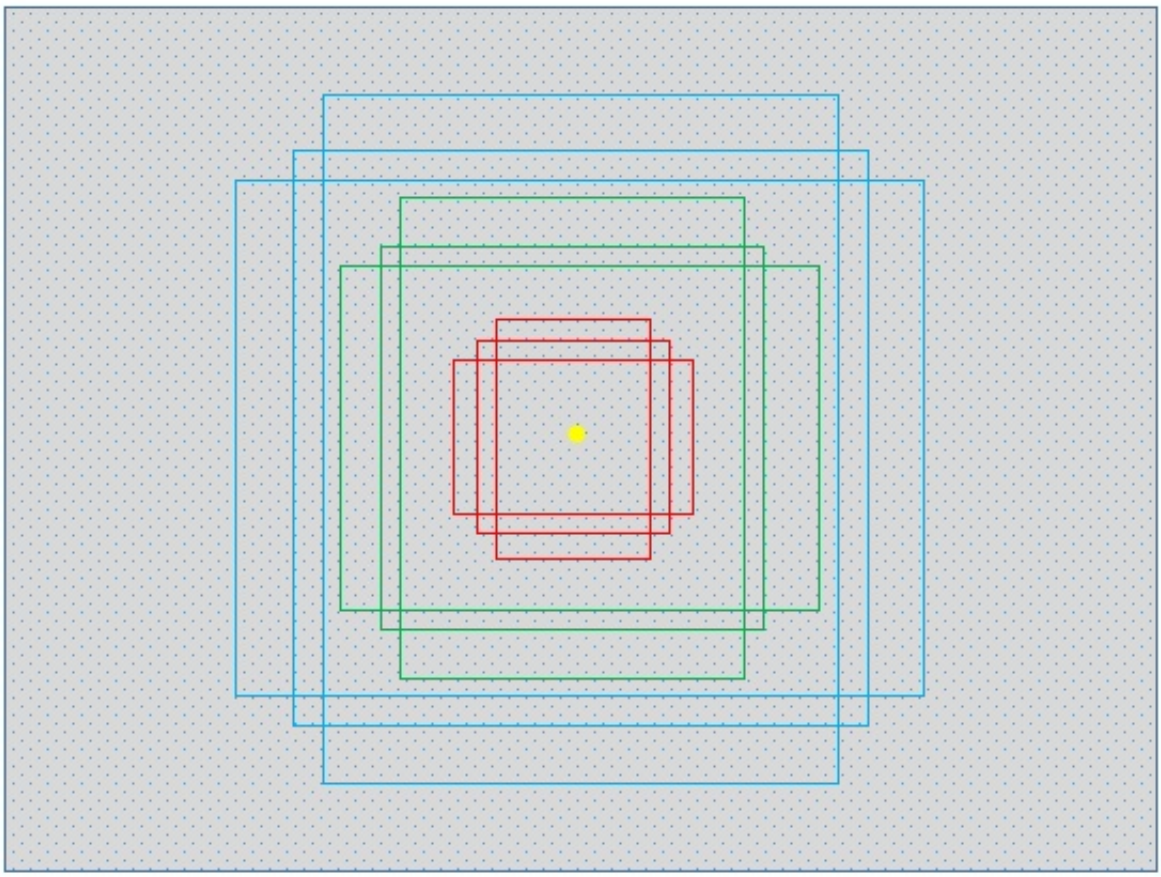
[参考]

- google - Understanding Faster-RCNN training input size
- github - Changing stride length decreased object detection accuracy
- 掘金 - 卷积神经网络模型解读汇总

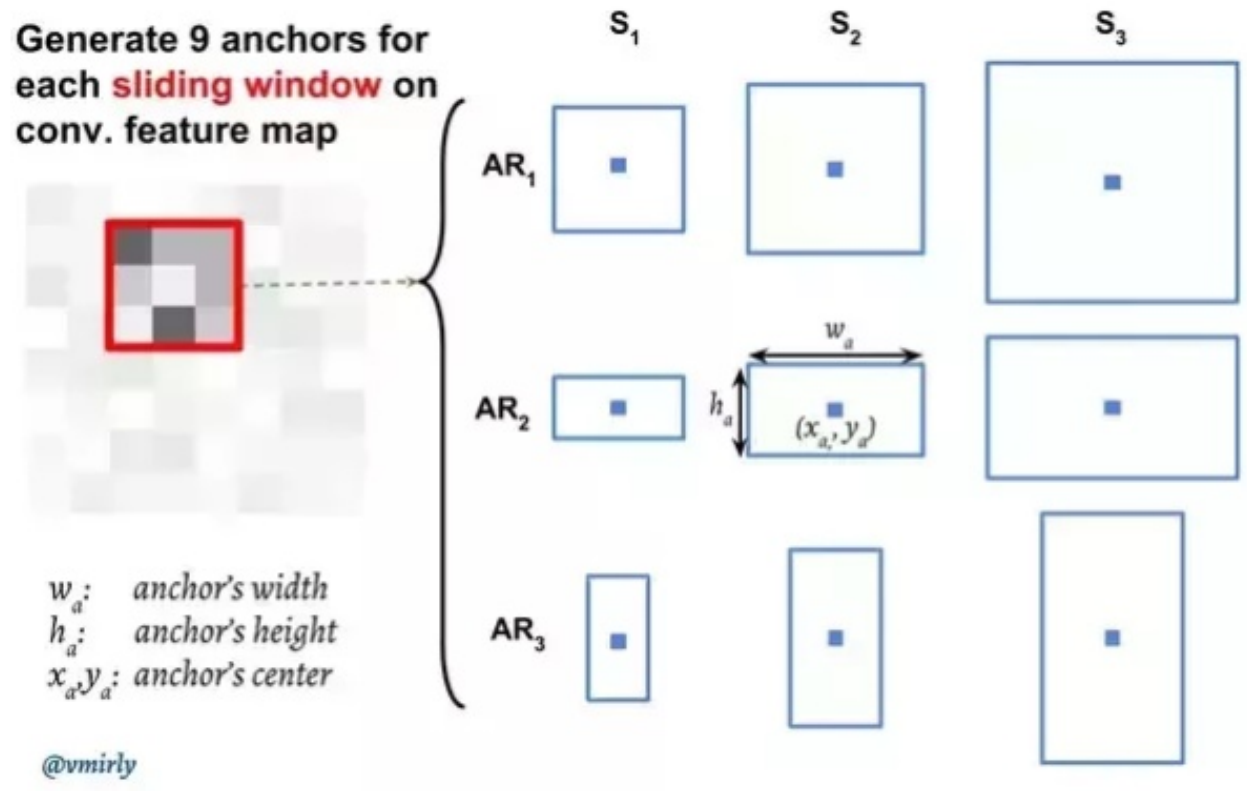
Anchor 的尺寸及其数量问题

尺寸问题

在 3.3 节的实现细节中，提到了生成 Anchor 的方法，采用三种尺寸 128^2、256^2、512^2，三种比例 1:2、1:1、2:1，如下图所示：



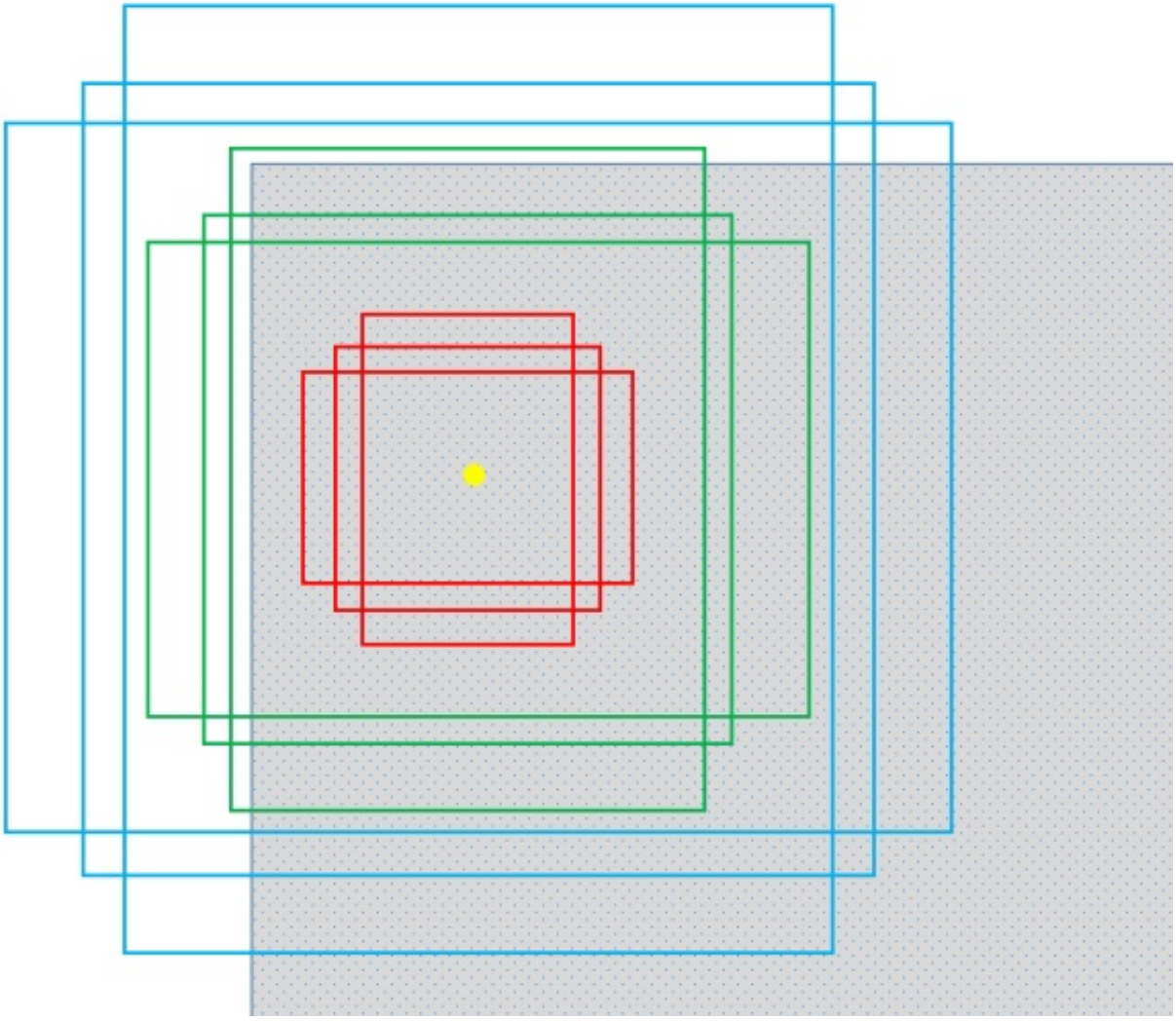
黄色的点即为中心点。或者是下图形式：



anchor 的数量问题

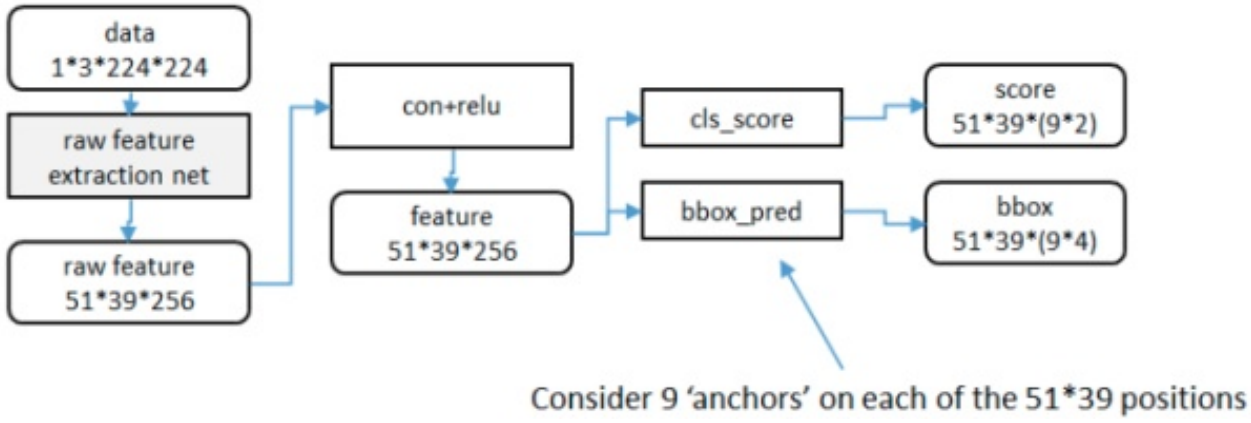
论文中也提到对于一个典型的 1000 ×600 的图片，生成的 anchor 的数量在 20k 左右，那么这个数字是怎么算出来的呢？ 首先需要算出来经过 ZFNet 或者 VGG16 之后，在最后一层得到的特征图的大小，因为总的 stride 为16，因此最终得到的特征图的大小为：Ceil（1000/16）× Ceil（600/16）≈ 63 × 38。这样经过 3×3 的网络之后就得到 63 × 38 × 9 = 21546 个 anchor，大约就是 20k。

但并不是产生的 anchor 都会给使用，这里需要处理掉跨越边界的 anchor box，如果在训练的时候不处理掉，会让训练难以收敛。



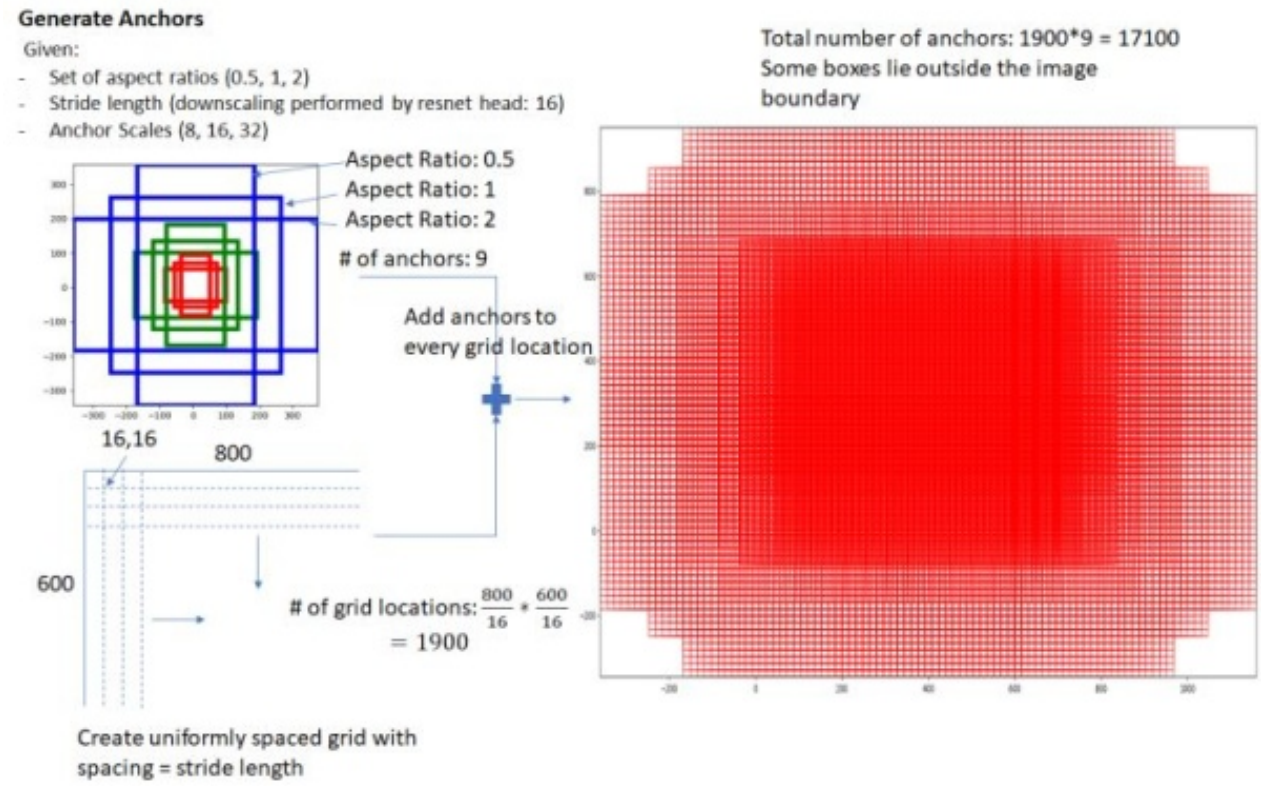
如上图，蓝色较大部分跨越了边界，绿色有少部分跨越，红色则没有跨越。但是，在测试阶段会保留所有的 proposal boxes ，但是会把跨越边界的部分裁剪掉。

从网上搜索资料的过程中，发现很多提到最终生成尺寸大小为 51×39×512（VGG16） 或者 51×39×512（ZFNet），如下面的一张示意图：



而且可以看到在很多的文章中都引用了这张图，但这里我就有点疑问，输入的是 1×3×224×224 得到的 raw feature 大小为什么是 51×39，得到的不应该是 (224/16)×(224/16) = 14×14?

其实这里隐含了一个信息，就是在输入到网络中的图片都需要进行缩放，重新缩放后的大小为 800 × 600。但这么计算下来的尺寸也应该是 (800/16)×(600/16) ≈ 50×38，为什么他们计算的就比这个多了一个像素呢，这里还真是没有搞明白，但我还是觉着应该是 50×38 。从下面的图中也可以印证自己的想法：



在 rbgirshick/py-faster-rcnn 给出了生成 anchor 的住要方法：

```
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
```

base_size：这里指的是总的 total stride，即通过 VGG16 或者 ZFNet之后的 total stride，也可以设置成其他的值，会影响到最终生成的 anchor 大小。但从Changing stride length decreased object detection accuracy 可以看到设置成较小的值，会让精度降低。

ratios：指的就是三种比率 1:2、1:1、2:1。

scales：其实就是 128^2、256^2、512^2，但这里的 scales 指的是在 feature map 上的大小，而不是在原图上的大小，可以看到 scales 的值为 8、16、32，通过乘上 base_size 就可以得到在原图上的大小 128、256、512。

运行代码可以得到下面的输出：

```
[[ -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.]]
```

9组数据表示 9 类 anchor，格式为 (x_1, y_1, x_2, y_2)，表示 anchor box 的左上角坐标和右下角坐标，比例大约在 {1:2、1:1、2:1}。

论文中，使用 ZFNet 以及最短边 s = 600 时，不同的尺度，及其宽高比的平均建议区域大小如下：

anchor	128^2, 2:1	128^2, 1:1	128^2, 1:2	256^2, 2:1	256^2, 1:1	256^2, 1:2	512^2, 2:1	512^2, 1:1	512^2, 1:2
proposal	188×111	113×114	70×92	416×229	261×284	174×332	768×437	499×501	355×715

还不是特别清楚上面的 proposal 是如何计算出来的？是对bbox Regression 之后的尺寸进行了平均的，还是在不同的尺寸图片上进行的平均？

经过查阅资料，上面得到的尺寸应该是通过 RPN 的 bbox regression 微调之后得到的平均值。

[参考]

- quora - How does the region proposal network (RPN) in Faster R-CNN work?
- medium - Faster R-CNN Explained
- 知乎 - faster rcnn中rpn的anchor，sliding windows，proposals?
- CSDN - Faster-RCNN算法精读

【参考汇总】

- cnblogs - Faster R-CNN论文详解
- cnblogs - Faster R-CNN
- 知乎 - 一文读懂Faster RCNN
- 个站 - Object Detection and Classification using R-CNNs
- CSDN - faster-rcnn 原理解析
- github - Why does it need a reshape layer in RPN's cls layer
- caffecn - py-faster rcnn中rpn的3x3的滑框用卷积层来定义的是为什么?
- google - Understanding Faster-RCNN training input size
- github - Changing stride length decreased object detection accuracy
- 掘金 - 卷积神经网络模型解读汇总
- quora - How does the region proposal network (RPN) in Faster R-CNN work?
- medium - Faster R-CNN Explained
- 知乎 - faster rcnn中rpn的anchor，sliding windows，proposals?
- CSDN - Faster-RCNN算法精读