ROI Pooling

- ***ROI***

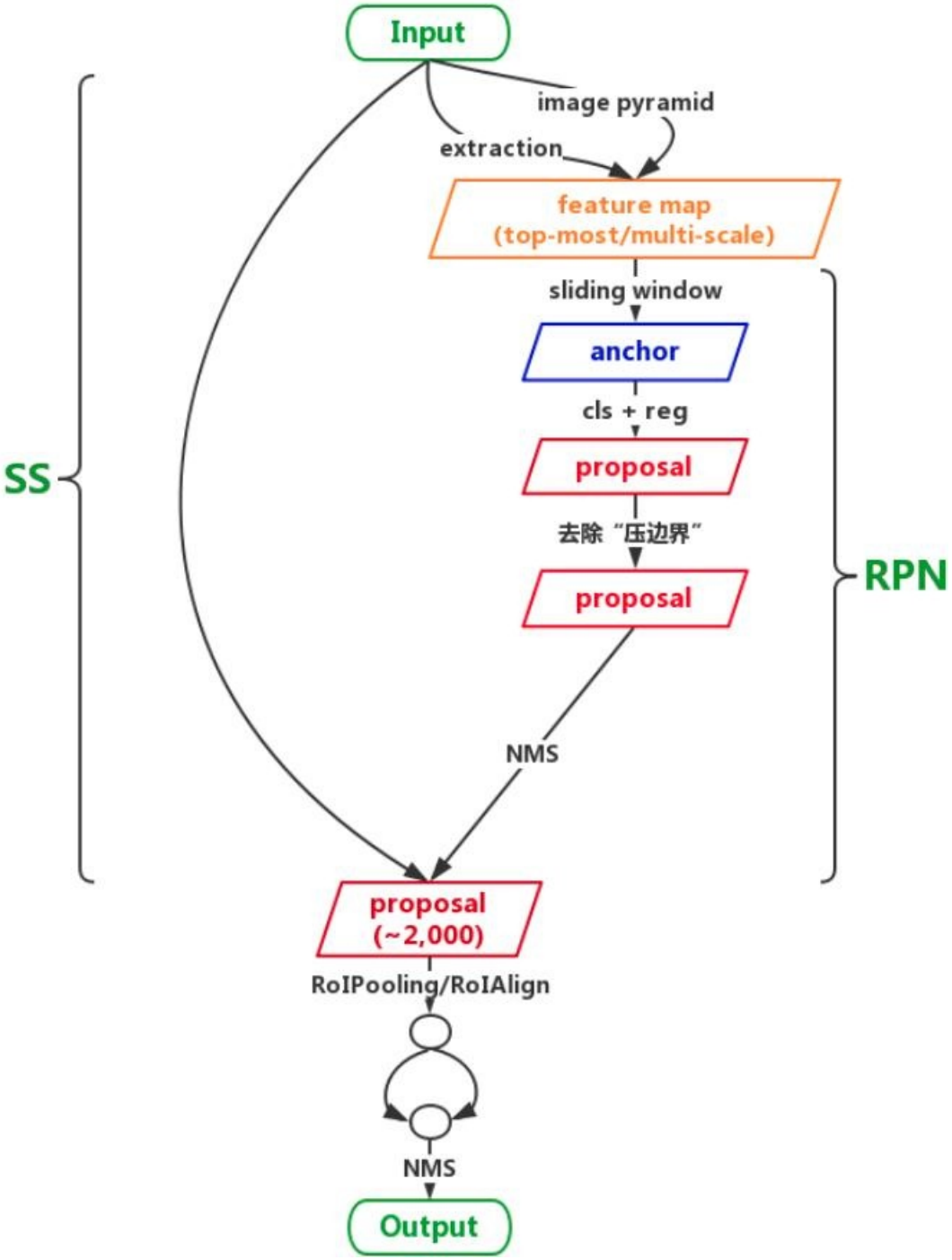
```
全称Region of Interest，特征图上的框;
1) 在Fast RCNN中，ROI是指Selective Search完成后得到的“候选框”在特征图上的映射
2) 在Faster RCNN中，候选框是经过RPN产生的，然后再把各个“候选框”映射到特征图上，得到ROIS
```

- ***ROI Pooling的输入***
  输入由两部分组成：
  1) 特征图：在Fast RCNN中，它位于ROI Pooling之前，在Faster RCNN中，它是与RPN共享那个特征图，通常我们称之为“share_conv”;
  2) rois:在Fast RCNN中，指的是Selective Search的输出;在Faster RCNN中指的是RPN的输出，一堆矩形候选框，形状为1×5×1×1(4个坐标+索引index)，其中值得注意的是，坐标的参考系不是针对feature map这张图的，而是针对原图的。
- ***ROI Pooling的输出***
  输出是batch个vector，其中batch的值等于ROI的个数，vector的大小为channel×w×h;ROI Pooling的过程就是一个个大小不同的box矩形框，都映射成大小固定(w×h)的矩形框;

RPN

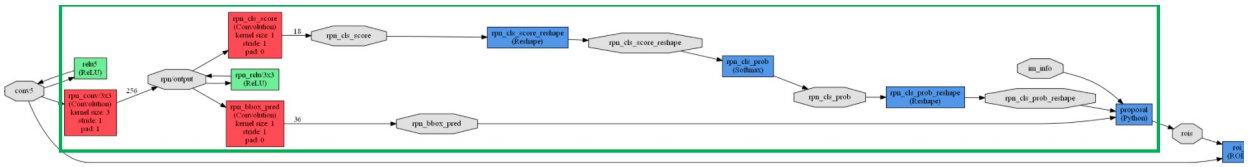
Region Proposal Network
本质是基于滑窗的无类别object检测器

- **RPN所在位置**

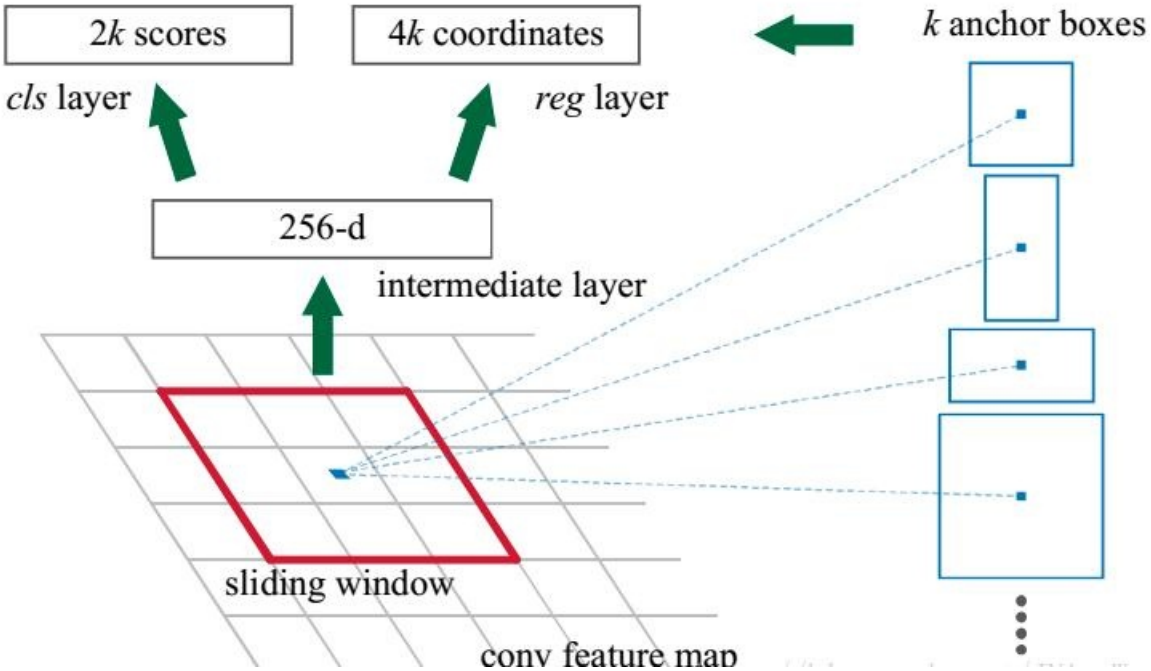


- **tip:**
  1) 只有在train时，cls+reg才能得到强监督信息(来源于ground truth)。即ground truth会告诉cls+reg结构，哪些才是真的前景，从而引导cls+reg结构学得正确区分前后景的能力；在reference阶段，就要靠cls+reg自力更生了。
  2) 在train阶段，会输出约2000个proposal，但只会抽取其中256个proposal来训练RPN的cls+reg结构；到了reference阶段，则直接输出最高score的300个proposal。此时由于没有了监督信息，所有RPN**并不知道这些proposal是否为前景**，整个过程只是惯性地推送一波无tag的proposal给后面的Fast R-CNN。
  3) RPN的运用使得region proposal的额外开销就只有一个两层网络。

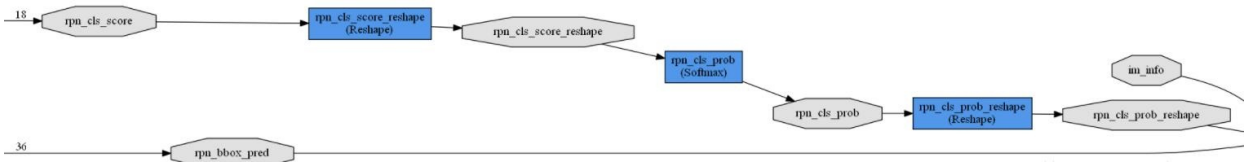
- **RPN的组成**



1) **RPN头部**，通过以下结构生成anchor(其实就是一堆有编号有坐标的bbox)



2) **RPN中部**，分类分支(cls)和边框回归分支(bbox reg)分别对这堆anchor进行计算



3) **RPN末端**

通过对 两个分支的结果进行汇总，来实现对anchor的 初步筛除（先剔除越界的anchor，再根据cls结果通过NMS算法去重）和 初步偏移（根据bbox reg结果），此时输出的都改头换面叫 Proposal 了

add:RPN之后，proposal成为ROI，被输入ROIPooling或ROIAlign中进行size上的归一化，**不属于RPN的范围**

tip:但是如果只在最后一层上feature map上映射回原图像，且初始产生的anchor被限定了尺寸下限，那么低于最小anchor尺寸的小目标虽然被anchor圈入，在后面的过程中依然容易被漏检。**FPN**的出现大大降低了小目标的漏检率，使得RPN如虎添翼。

- anchor机制

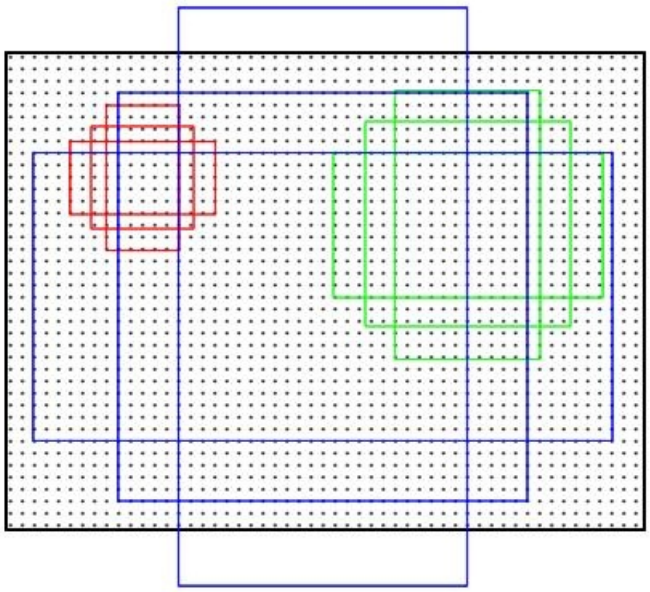
  rpn网络的核心，anchor给出一个基准窗大小，按照倍数和长宽得到不同大小的窗，论文中基准窗的大小为16,给了(8,16,32)三种倍数和(0.5,1,2)三种比例，这样能够得到一共9种尺度的anchor，在对60×40的map进行滑窗时(步长为1)，以中心像素为基点构造9种anchor映射到原来的1000×600图像中，映射比例为16倍。那么总共可以得到60×40×9大约2万个anchor。

- 训练

  RPN网络训练，那么就涉及ground truth和loss function的问题。对于左支路，ground truth为anchor是否为目标，用0/1表示。那么怎么判定一个anchor内是否有目标呢？论文中采用了这样的规则：1）假如某anchor与任一目标区域的IoU最大，则该anchor判定为有目标；2）假如某anchor与任一目标区域的

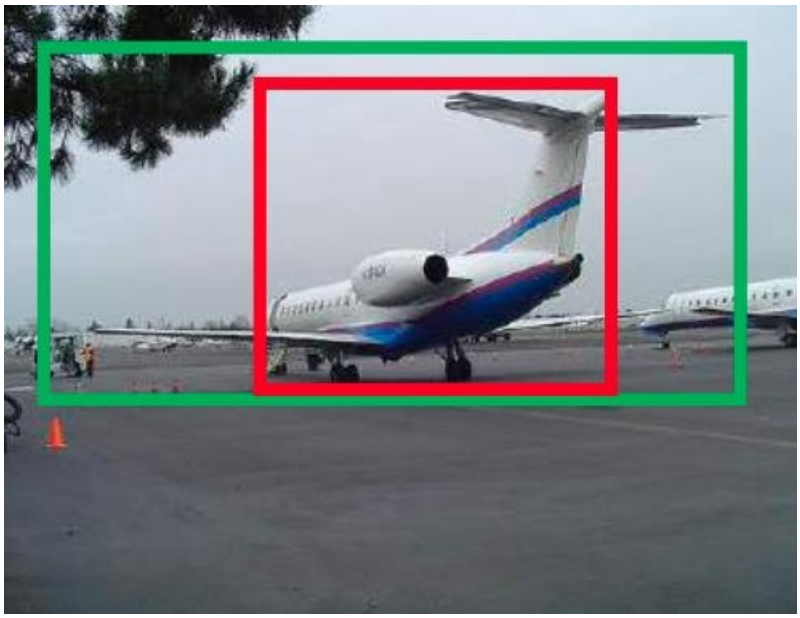
IoU>0.7，则判定为有目标；3）假如某anchor与任一目标区域的IoU<0.3，则判定为背景。所谓IoU，就是预测box和真实box的覆盖率，其值等于两个box的交集除以两个box的并集。其它的anchor不参与训练。

- 联合训练
  四步训练法：
  1) 单独训练RPN网络，网络参数由预训练模型载入
  2) 单独训练Fast-RCNN网络，将第一步RPN的输出候选区域作为检测网络的输入。具体而言，RPN输出一个候选框，通过候选框截取原图像，并将截取后的图像通过几次conv-pool，然后再通过roi-pooling和fc再输出两条支路，一条是目标分类softmax，另一条是bbox回归。截止到现在，两个网络并没有共享参数，只是分开训练了;
  3) 再次训练RPN，此时固定网络公共部分的参数，只更新RPN独有部分的参数；
  4) 拿RPN的结果再次微调Fast-RCNN网络，固定网络公共部分的参数，只更新Fast-RCNN独有部分的参数。

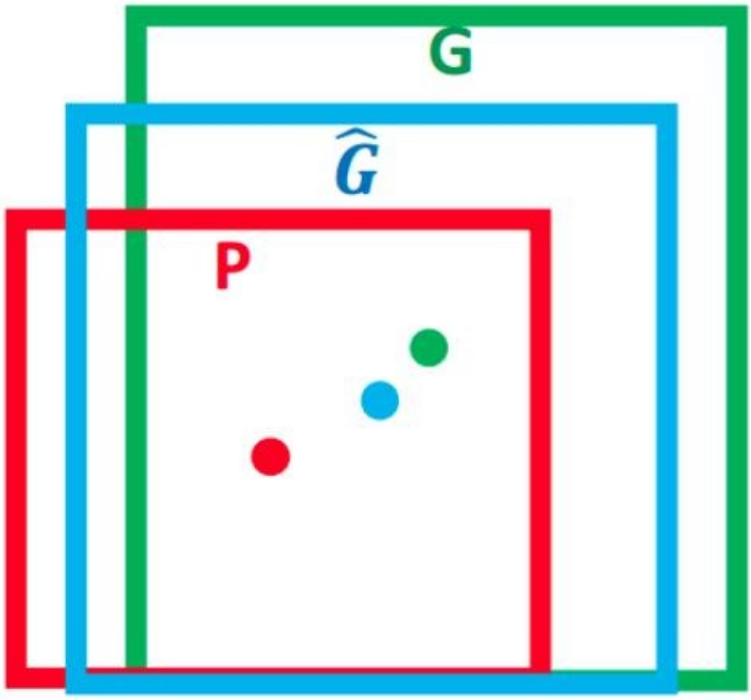


BBR

Bounding Box Regression
边框回归



对于上图，绿色的框代表Ground Truth，红色的框为Selective Search提取的Region Proposal。即便红色的框被分类器识别为飞机，但是由于红色的框定位不准(IOU<0.5)，那么这张图片相当于没有正确的检测出飞机，需要对框进行微调。



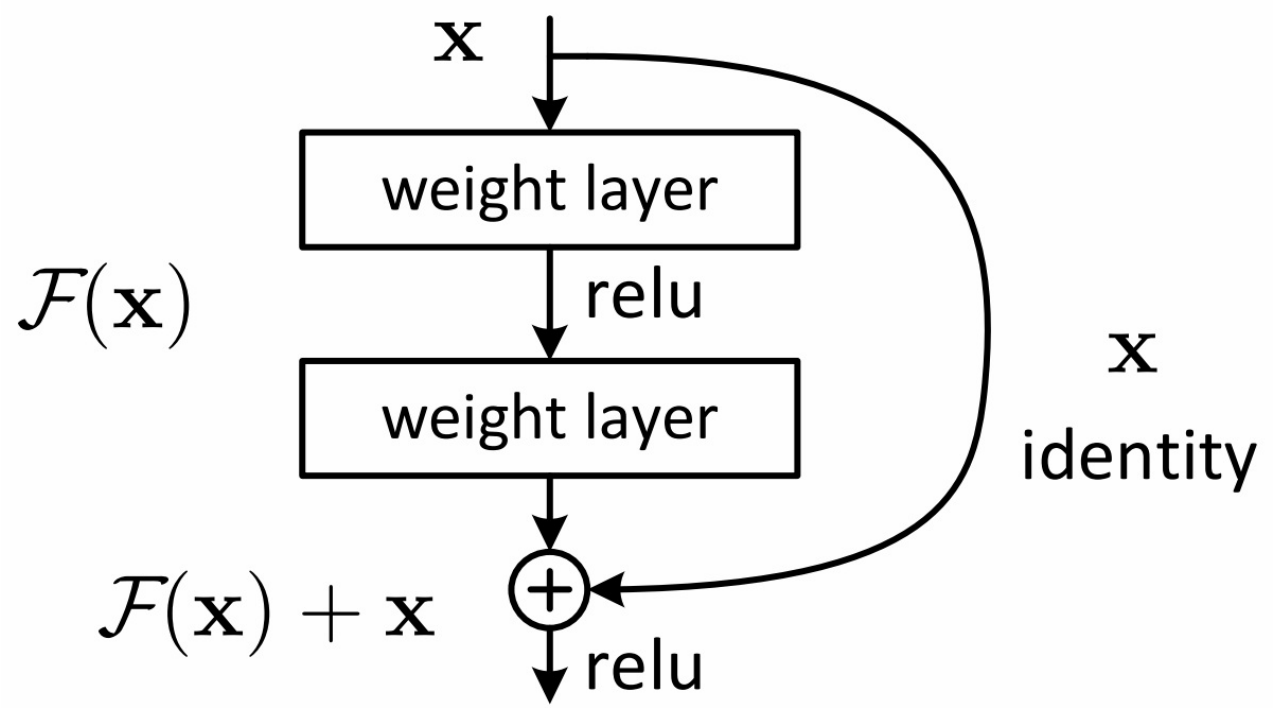
窗口使用四维向量(x,y,w,h)来表示，目标是寻找一种关系使得输入原始的窗口P经过映射得到一个跟真实窗口G更接近的回归窗口G^。边框回归的目的既是：给定(Px,Py,Pw,Ph)寻找一种映射f，使得
$$f(Px,Py,Pw,Ph) = (Ĝx,Ĝy,Ĝw, Ĝh)$$
并且$$(Ĝx,Ĝy,Ĝw, Ĝh) ≈ (Gx,Gy,Gw, Gh)$$

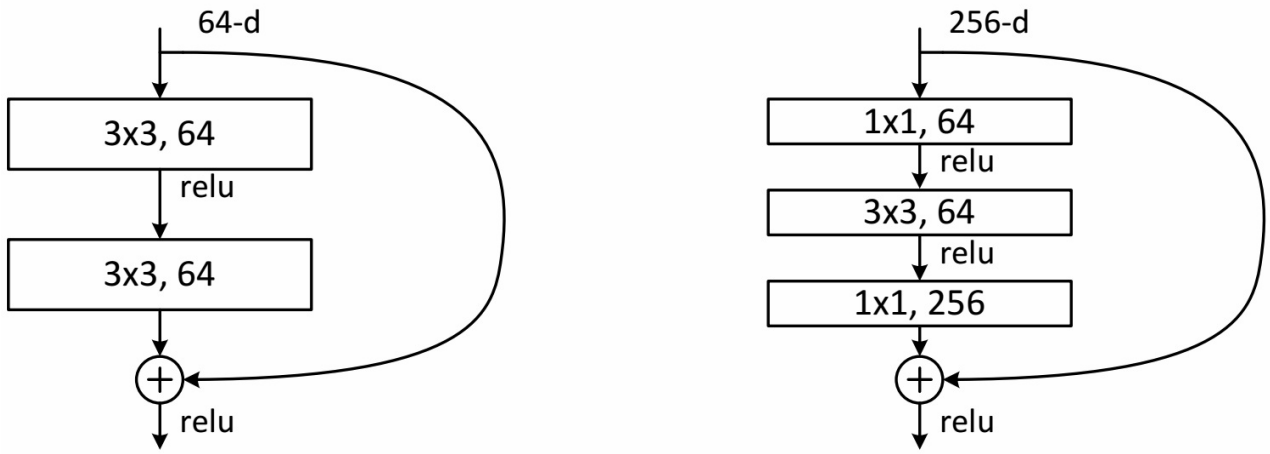
- ***步骤***
  平移+尺度放缩
  1) 先做平移(Δx,Δy)， Δx=Pwdx(P), Δy=Phdy(P) 这是R-CNN论文的：
  $$Ĝx=Pwdx(P)+Px,(1) $$
  $$Ĝ y=Phdy(P)+Py,(2) $$
  2) 然后再做尺度缩放(Sw,Sh), Sw=exp(dw(P)), Sh=exp(dh(P)), 对应论文中：
  $$Ĝ w=Pwexp(dw(P)),(3) $$
  $$Ĝ h=Phexp(dh(P)),(4) $$
  边框回归学习的就是dx(P),dy(P),dw(P),dh(P)这四个变换
- Input
  RegionProposal→P=(Px,Py,Pw,Ph)RegionProposal→P=(Px,Py,Pw,Ph) ，这个是什么？ 输入就是这四个数值吗？其实真正的输入是这个窗口对应的 CNN 特征，也就是 R-CNN 中的 Pool5 feature（特征向量）。 (注：训练阶段输入还包括 Ground Truth， 也就是下边提到的t∗=(tx,ty,tw,th)t∗=(tx,ty,tw,th))
- output
  需要进行的平移变换和尺度缩放 dx(P),dy(P),dw(P),dh(P)dx(P),dy(P),dw(P),dh(P)， 或者说是Δx,Δy,Sw,ShΔx,Δy,Sw,Sh 。 我们的最终输出不应该是 Ground Truth 吗？ 是的， 但是有了这四个变换我们就可以直接得到 Ground Truth， 这里还有个问题， 根据(1)(4)我们可以知道， P 经过 dx(P),dy(P),dw(P),dh(P)dx(P),dy(P),dw(P),dh(P) 得到的并不是真实值 G， 而是预测值Ĝ G^。 的确， 这四个值应该是经过 Ground Truth 和 Proposal 计算得到的真正需要的平移量(tx,ty)(tx,ty) 和尺度缩放(tw,th)(tw,th) 。
  这也就是 R-CNN 中的(6)(9)：
  $$ tx=(Gx−Px)/Pw,(6) $$
  $$ ty=(Gy−Py)/Ph,(7) $$
  $$ tw=log(Gw/Pw),(8) $$

$$ th=log(Gh/Ph),(9) $$
那么目标函数可以表示为 d∗(P)=wT∗Φ5(P)d∗(P)=w∗TΦ5(P)， Φ5(P)Φ5(P)是输入 Proposal 的特征向量，w∗w∗是要学习的参数（*表示 x,y,w,h， 也就是每一个变换对应一个目标函数） , d∗(P)d∗(P) 是得到的预测值。 我们要让预测值跟真实值t∗=(tx,ty,tw,th)t∗=(tx,ty,tw,th)差距最小。

ResNet

深度残差网络





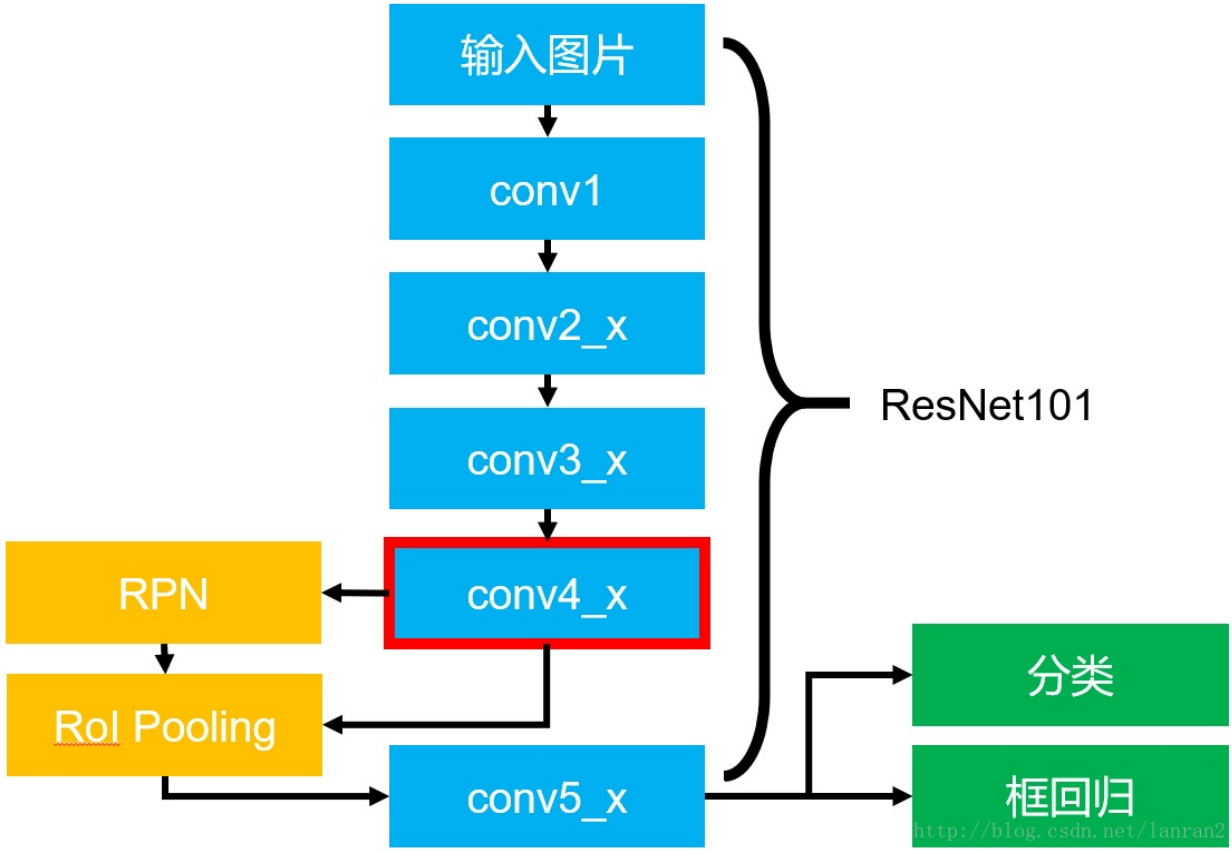
identity mapping “弯弯的曲线” residual mapping “除曲线那部分”。

- 两种shortcut Connection方式;
- 5种深度的ResNet，分别为18,34,50,101,152

```
所有的网络都分成5部分，分别为:conv1,conv2_x,conv3_x,conv4_x,conv5_x
以101-layer为例，首先输出7×7×64的卷积，然后经过3+4+23+3 = 33 个building block，每个block为3层，所以33×3 =
99 层，最后有个fc层(用于分类)，所以1+99+1 = 101，
tip：101仅指卷积或全连接层，而激活层或者Pooling层并没有计算在内。
50-layer和101-layer不同在于conv4_x，ResNet50有6个block，而ResNet101有23个block，差了17个block，也就是17×
```

```
3 = 51 层。
```

- 基于ResNet101的Faster RCNN



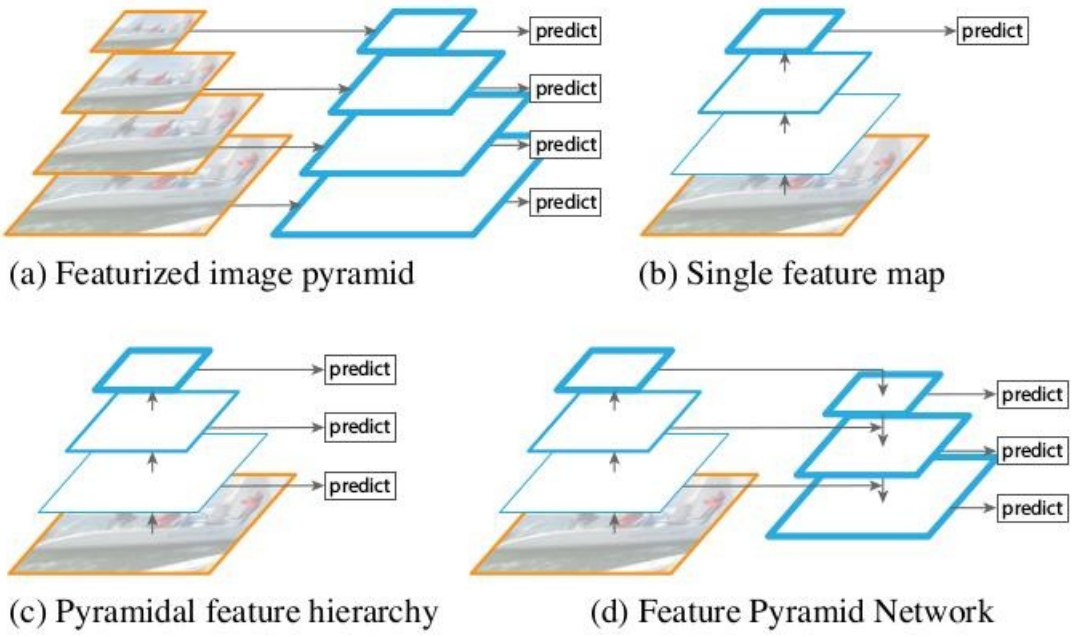
蓝色的部分为ResNet101，可以发现con4_x的最后的输出为RPN和ROI Pooling的共享部分，而conv5_x(共9层网络)都作用于ROI Pooling之后的一堆特征图(14×14×1024)，特征图的大小维度也刚好符合原本的ResNet101中conv5_x的输入;一定要记得最后接一个average pooling，得到2048维特征，分别用于分类和框回归。

FPN

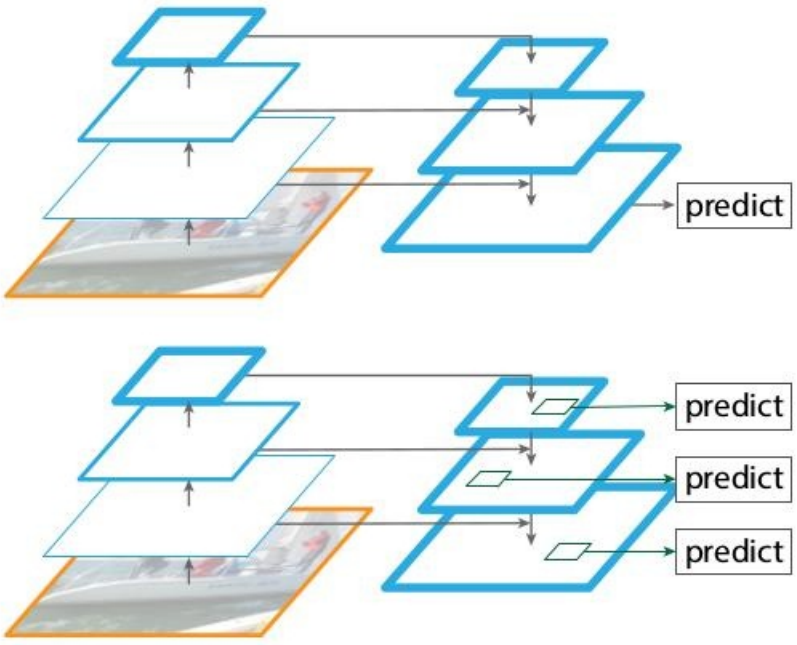
Feature Pyramid Networks

> 低层的特征语义信息比较少，但是目标位置准确；高层的特征语义信息比较丰富，但是目标位置比较粗略。另外虽然也有些算法采用多尺度特征融合的方式，但是一般是采用融合后的特征做预测，而本文不一样的地方在于预测是在不同特征层独立进行的。

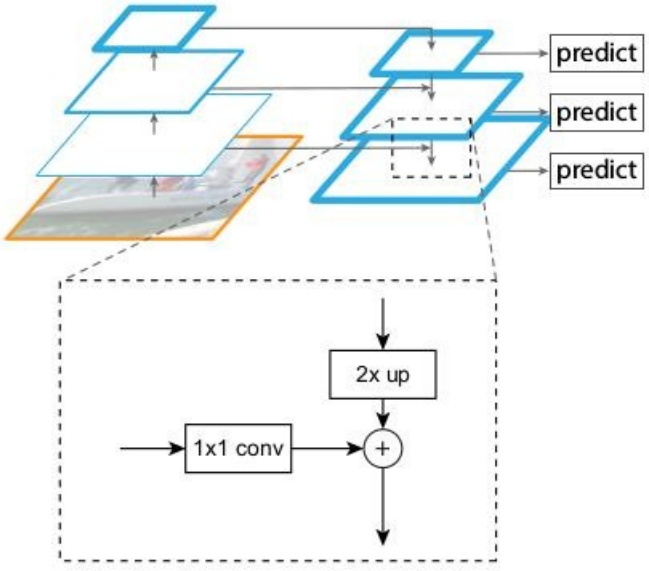
- **四种利用特征的形式**



(a) 图像金字塔，即将图像做成不同的scale，然后不同scale的图像生成对应的不同scale的特征。
(b) 像SPP net，Fast RCNN，Faster RCNN是采用这种方式，即仅采用网络最后一层的特征。
(c) 像SSD（Single Shot Detector）采用这种多尺度特征融合的方式，没有上采样过程，即从网络不同层抽取不同尺度的特征做预测，这种方式不会增加额外的计算量。作者认为SSD算法中没有用到足够低层的特征（在SSD中，最低层的特征是VGG网络的conv4_3），而在作者看来足够低层的特征对于检测小物体是很有帮助的。
(d) 本文作者是采用这种方式，顶层特征通过上采样和低层特征做融合，而且每层都是独立预测的。



上面一个带有skip connection的网络结构在预测的时候是在finest level（自顶向下的最后一层）进行的，简单讲就是经过多次上采样并融合特征到最后一步，拿最后一步生成的特征做预测。而下面一个网络结构和上面的类似，区别在于预测是在每一层中独立进行的。

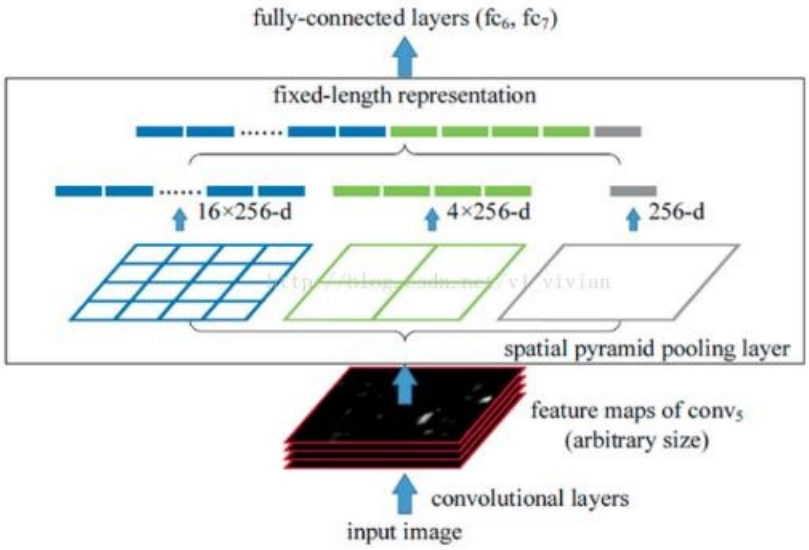


作者的主网络采用ResNet，算法大致结构如上图，**一个自低向上的线路，一个自顶向下的线路，横向连接(lateral connection)**，图中放大的区域就是横向连接，这里的1×1卷积核的主要作用是减少卷积核的个数，也就是减少feature map的个数，并不改变feature map的尺寸大小。

自底向上其实就是网络的前向过程。

SppNet

Spatial Pyramid Pooling 空间金字塔池化
全连接层需要指定输入层和输出层神经元个数，所以需要规定输入的feature的大小，导致神经网络需要输入经过crop、warp操作的固定尺寸的图片。
ORI: images –> crop/warp –> conv layers –> fc layers –> output
SPP: images –> conv layers –> Spatial Pyramid Pooling –> fc layers –> output
- **整体过程**



黑色图片代表卷积之后的特征图，接着我们以不同大小的块来提取特征，分别是4×4,2×2,1×1，放到特征图上就可以得到16+4+1=

21种不同大小的块(Spatial bins)，从这21个块中，每个块提取一个特征，这样刚好就是我们要提取的21维特征向量，这种以不同的大小格子的组合方式来池化的过程就是空间金字塔池化(SPP)；
输出向量大小为MK，M=*#bins，K=#filters，作为全连接层的输入，例如上图，conv5计算出的feature map是任意大小的，经过SPP后，就可以变成固定大小的输出了，上图为(16+4+1)×256的特征传入fc。*

RetinaNet

detector主要分为以下两大门派：
one stage–> YOLOV1–3,SSD 精度低但速度快
two stage–> R-CNN,SPPNet,Fast/Faster R-CNN 精度高但速度慢

1) one stage受制于‘类别不平衡’(检测算法早期会生成一大波bbox，绝大多数属于负样本background，即使分类器无脑地把所有的bbox同一归类为background，accuracy也可以刷得很高，于是分类器的训练就失败了)，交叉熵损失(CE)无法抗衡“类别极不平衡”。
2) two stage有RPN，第一阶段的RPN会对anchor进行简单的二分类(只是简单的区分是前景还是背景，并不区别究竟属于那个类)；第二阶段分类器登场，在初筛过后的bbox上进行难度小得多的第二波分类(细分类)

- focal loss

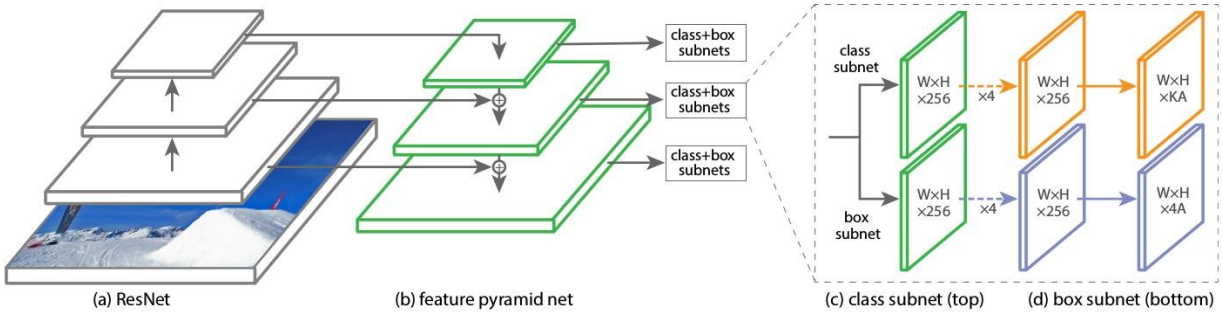
  focal loss的标准公式非常简单：

$$FL(p_t) = -(1-p_t)^{\gamma}\log(p_t) = (1-p_t)^{\gamma}CE(\hat{y})_i$$

  也可以更复杂一点（论文中的实验即采用此公式）：

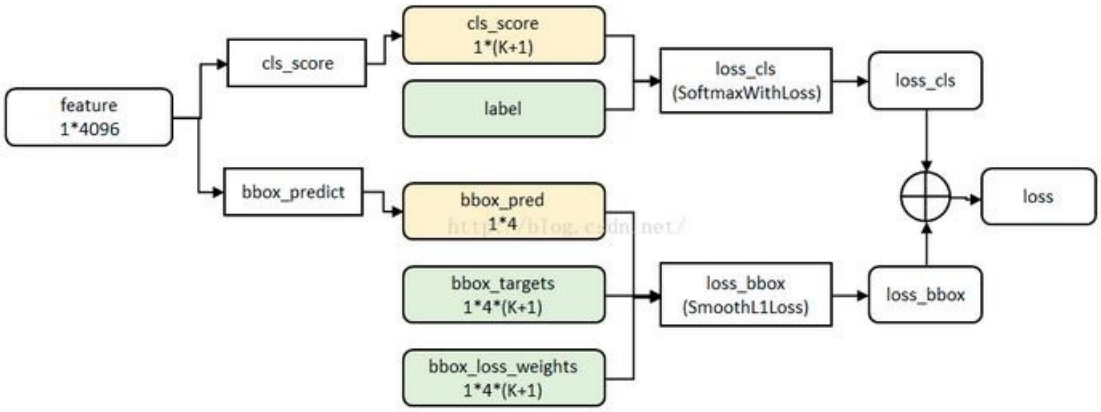
$$FL(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t)$$

  在原本的交叉熵损失函数前乘上一个权重，实验中发现γ=2，α=0.25的取值组合效果最好。
- RetinaNet:



FasterRCNN中的SmoothL1Loss

- **Fast RCNN损失函数，多任务损失**



multi-task数据结构

fast rcnn网络有两个同级输出层(cls score和bbox_predict层)，都是全连接层，称为multi-task
1) clsscore层，用于分类，输出k+1维数组p，表示属于k类和背景的概率。对每个ROI输出离散型概率分布：

$p = (p_0, p_1, \cdots, p_k)$，通常，p由k+1类的全连接层利用softmax计算得出。

2) bbox_predict层，用于调整候选区域位置，输出bounding box回归的偏移，输出4*k维数组t，表示分别属于k类时，应该平移缩放的参数：

$t^k = (t_x^k, t_y^k, t_w^k, t_h^k)$

3) loss_cls层评估分类损失函数，loss_bbox评估检测框定位的损失函数，比较真实分类对应的预测平移缩放参数和真实平移缩放参数的参别：

$$L_{\text{loc}}(t^u, v) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(t_i^u - v_i), \qquad (2)$$

in which

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases} \qquad (3)$$

使得loss对于离群点更加鲁棒。
最后总损失为两者加权和，如果分类为背景则不考虑定位损失。

$$L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \lambda[u \geq 1] L_{\text{loc}}(t^u, v), \qquad (1)$$

艾弗森括号指数函数[u≥1]表示背景候选区域即负样本不参与回归损失，不需要对候选区域进行回归操作。λ控制分类损失和回归损失的平衡。Fast R-CNN论文中，所有实验λ=1。

- **Faster RCNN损失函数**

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

其中：

p_i为 anchor 预测为目标的概率;

GT标签：$p_i^* = \begin{cases} 0 & negative\ label \\ 1 & positive\ label \end{cases}$；

$t_i = \{t_x, t_y, t_w, t_h\}$是一个向量，表示预测的 bounding box 包围盒的 4 个参数化坐标;

t_i^*是与 positive anchor 对应的 ground truth 包围盒的坐标向量;

$L_{cls}(p_i, p_i^*)$是两个类别（目标 vs.非目标）的对数损失：

$$L_{cls}(p_i, p_i^*) = -\log\left[p_i^* p_i + (1-p_i^*)(1-p_i)\right]$$

$L_{reg}(t_i, t_i^*)$是回归损失，用$L_{reg}(t_i, t_i^*) = R(t_i - t_i^*)$来计算，R 是 smooth L1 函数。

$p_i^* L_{reg}$这一项意味着只有前景 anchor（$p_i^* = 1$）才有回归损失，其他情况就没有（$p_i^* = 0$）。cls 层和 reg 层的输出分别由$\{p_i\}$和$\{u_i\}$组成，这两项分别由N_{cls}和N_{reg}以及一个平衡权重λ归一化（早期实现及公开的代码中，$\lambda = 10$，cls 项的归一化值为 mini-batch 的大小，即$N_{cls} = 256$，reg 项的归一化值为 anchor 位置的数量，即N_{reg}~2400（$40 * 60$），这样 cls 和 reg 项差不多是等权重的。

SmoothL1LossLayer 计算一张图片的损失函数

$$\lambda \frac{1}{N_{reg}} \sum_i {p_i}^* L_reg(t_i, {t_i}^*) \tag{1}$$

- i ： mini-batch 的 anchor 的索引。
- p_i ：目标的预测概率。
- ${p_i}^*$ ：target二分类是否有物体，有物体为1，否则为0。
- t_i 是一个四点向量，预测坐标
- ${t_i}^*$ 是一个四点向量，是ground truth boungding box的坐标（真实坐标）

$$L_{reg}(t,t_i^*) = R(t_i - t_i*) \tag{2}$$

bottom[0]预测坐标，即 t_i
bottom[1]target坐标，即${t_i}^*$
bottom[2]inside，有物体，即有前景(foreground)时为1，否则为0，即${p_i}^*$
bottom[3]outside，没有前景（fg）也没有后景（bg）的为0，其他为1/（bg+fg），对应于加号右边的系数部分。

Lreg的公式如下，其中 $x = t_i - t_i*$，

$$smooth_{L1}(x) = \begin{cases} 0.5x^2, if\,|x| < 1 \\ |x| - 0.5, otherwise \end{cases} \tag{3}$$

$p_i * L_{reg}(t_i, {t_i}^*)$表明只有有fg（20个物体类别）的才有regression loss.

SmoothL1LossLayer 计算一张图片的损失函数

$$\lambda \frac{1}{N_{reg}} \sum_i {p_i}^* L_r eg(t_i, {t_i}^*) \tag{1}$$

- i ： mini-batch 的 anchor 的索引。
- p_i ：目标的预测概率。
- ${p_i}^*$ ：target二分类是否有物体，有物体为1，否则为0。
- t_i 是一个四点向量，预测坐标
- ${t_i}^*$ 是一个四点向量，是ground truth boungding box的坐标（真实坐标）

$$L_{reg}(t, t_i^*) = R(t_i - t_i*) \tag{2}$$

bottom[0]预测坐标，即 t_i
bottom[1]target坐标，即${t_i}^*$
bottom[2]inside，有物体，即有前景(foreground)时为1，否则为0，即${p_i}^*$
bottom[3]outside，没有前景（fg）也没有后景（bg）的为0，其他为1/（bg+fg），对应于加号右边的系数部分。

Lreg的公式如下，其中 $x = t_i - t_i*$ ，

$$smooth_{L1}(x) = \begin{cases} 0.5x^2, if\,|x| < 1 \\ |x| - 0.5, otherwise \end{cases} \tag{3}$$

$p_i * L_{reg}(t_i, {t_i}^*)$表明只有有fg（20个物体类别）的才有regression loss.

GIOU

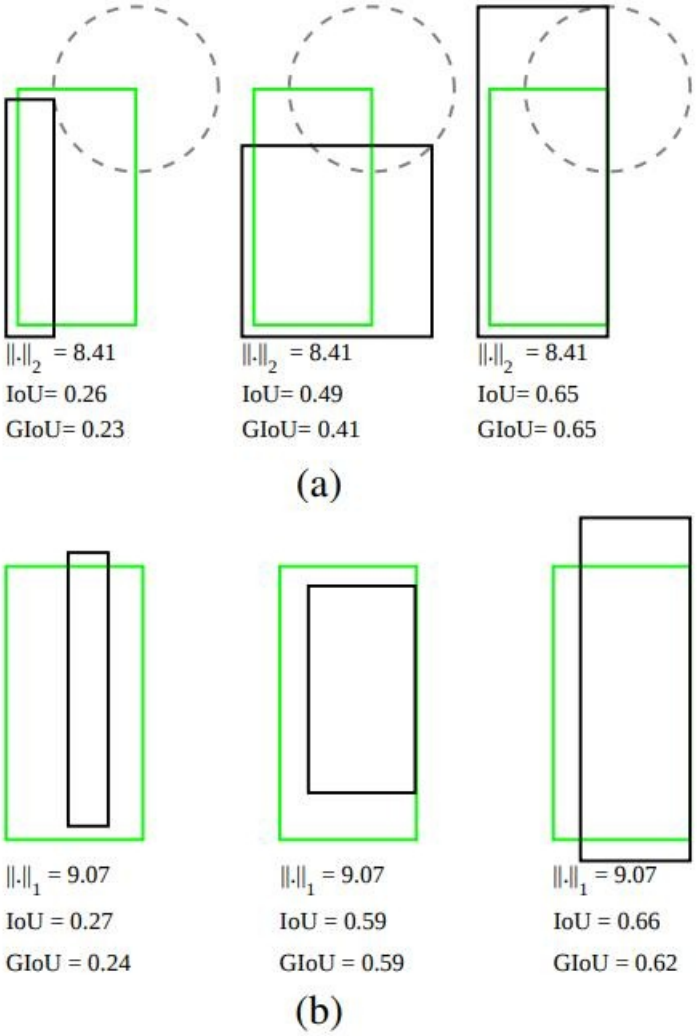
- ***IOU作为损失存在的问题***
  1) 如果两个对象不重叠，IOU值将为零，不论相差多远，即不会反映两个形状彼此之间的距离，如果用IOU用作损耗，则其梯度将为零，无法优化。
  2) IOU的值无法反应两个对象之间如何重叠，同一个IOU值两个对象间有多种重叠方式。
- ***GIOU计算***

Algorithm 1: Generalized Intersection over Union

input : Two arbitrary convex shapes: $A, B \subseteq \mathbb{S} \in \mathbb{R}^n$
output: $GIoU$

1 For A and B, find the smallest enclosing convex object C, where $C \subseteq \mathbb{S} \in \mathbb{R}^n$
2 $IoU = \frac{|A \cap B|}{|A \cup B|}$
3 $GIoU = IoU - \frac{|C \backslash (A \cup B)|}{|C|}$

- ***IOU与GIOU对比***



- ***IOU与GIOU loss计算过程***

Algorithm 2: IoU and $GIoU$ as bounding box losses

input : Predicted B^p and ground truth B^g bounding box coordinates:

$$B^p = (x_1^p, y_1^p, x_2^p, y_2^p), \quad B^g = (x_1^g, y_1^g, x_2^g, y_2^g).$$

output: $\mathcal{L}_{IoU}$, $\mathcal{L}_{GIoU}$.

For the predicted box B^p, ensuring $x_2^p > x_1^p$ and $y_2^p > y_1^p$:

$$\hat{x}_1^p = \min(x_1^p, x_2^p), \quad \hat{x}_2^p = \max(x_1^p, x_2^p),$$
$$\hat{y}_1^p = \min(y_1^p, y_2^p), \quad \hat{y}_2^p = \max(y_1^p, y_2^p).$$

Calculating area of B^g: $A^g = (x_2^g - x_1^g) \times (y_2^g - y_1^g)$.

Calculating area of B^p: $A^p = (\hat{x}_2^p - \hat{x}_1^p) \times (\hat{y}_2^p - \hat{y}_1^p)$.

Calculating intersection $\mathcal{I}$ between B^p and B^g:

$$x_1^{\mathcal{I}} = \max(\hat{x}_1^p, x_1^g), \quad x_2^{\mathcal{I}} = \min(\hat{x}_2^p, x_2^g),$$
$$y_1^{\mathcal{I}} = \max(\hat{y}_1^p, y_1^g), \quad y_2^{\mathcal{I}} = \min(\hat{y}_2^p, y_2^g),$$
$$\mathcal{I} = \begin{cases} (x_2^{\mathcal{I}} - x_1^{\mathcal{I}}) \times (y_2^{\mathcal{I}} - y_1^{\mathcal{I}}) & \text{if} \quad x_2^{\mathcal{I}} > x_1^{\mathcal{I}}, y_2^{\mathcal{I}} > y_1^{\mathcal{I}} \\ 0 & \text{otherwise.} \end{cases}$$

Finding the coordinate of smallest enclosing box B^c:

$$x_1^c = \min(\hat{x}_1^p, x_1^g), \quad x_2^c = \max(\hat{x}_2^p, x_2^g),$$
$$y_1^c = \min(\hat{y}_1^p, y_1^g), \quad y_2^c = \max(\hat{y}_2^p, y_2^g).$$

Calculating area of B^c: $A^c = (x_2^c - x_1^c) \times (y_2^c - y_1^c)$.

$IoU = \frac{\mathcal{I}}{\mathcal{U}}$, where $\mathcal{U} = A^p + A^g - \mathcal{I}$.

$GIoU = IoU - \frac{A^c - \mathcal{U}}{A^c}$.

$\mathcal{L}_{IoU} = 1 - IoU, \quad \mathcal{L}_{GIoU} = 1 - GIoU.$

RefineDet

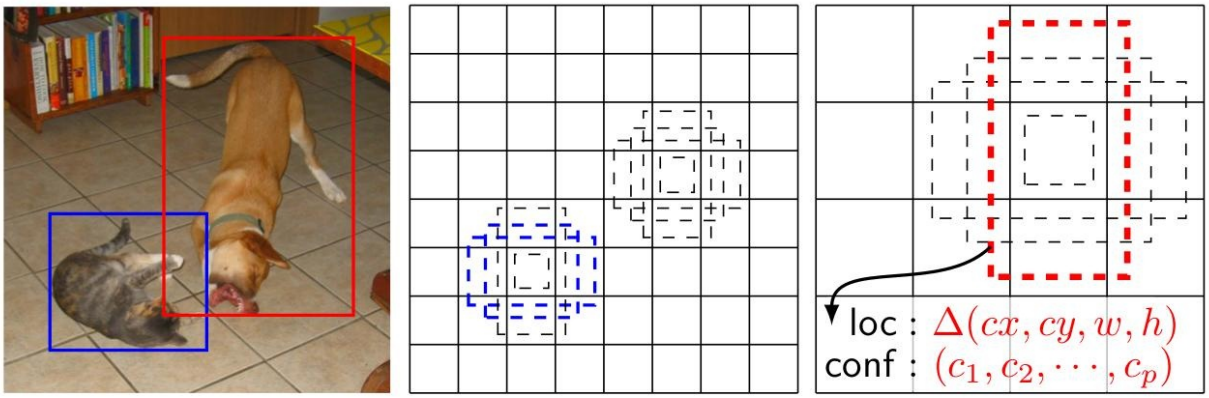
Single-Shot Refinement Neural Network for Object Detection(CVPR2018)

SSD

single shot multibox detector

- **本文提出的SSD算法是一种直接预测bounding box的坐标和类别的object detection算法，没有生成proposal的过程**
- 基本的网络结构是基于VGG16，在ImageNet数据集上预训练完以后用两个新的卷积层代替fc6和fc7，另外对pool5也做了一点小改动，还增加了4个卷积层构成本文的网络。
  1) 文章的一个核心是作者同时采用lower和upper的feature maps做检测
  2) default box，是指在feature map的每个小格(cell)上都有一系列固定大小的box，如下图有4个(虚线框)，假设每个feature cell有k个default box，那么对与每个default box都需要预测c个类别score和offset，那么如果一个feature map的大小是m×n，也就是有m×n个feature map cell，那么这个feature map就一共有k×m×n个default box，每个default box都需要预测4个坐标相关的值和c+1个类别概率(**实**

际code中分别用不同数量的3×3卷积核对该层feature map进行卷积，不如卷积核数量为(c+1)×k对应confidence输出，表示每个default box的confidence，就是类别；卷积核数量为4×k对应localization输出，表示每个default box的坐标)，default box与Faster RCNN中anchor很像，Faster RCNN只用在最后一个卷积层，但SSD中用在多个不同层的feature map上。**下图还有个重要信息：训练阶段，算法在一开始会先将这些default box和ground truth box进行匹配，比如蓝色的两个虚线框和猫的ground truth box匹配上了，即一个ground truth可能对应多个default box;在预测阶段，直接预测每个default box的偏移以及每个类别相应的得分，最后通过NMS得到最终的结果**



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

- **default box的scale(大小)和aspect ratio(纵横比)**

  1) scale计算公式：

$$s_k = s_{\min} + \frac{s_{\max} - s_{\min}}{m - 1}(k - 1), \quad k \in [1, m] \tag{4}$$

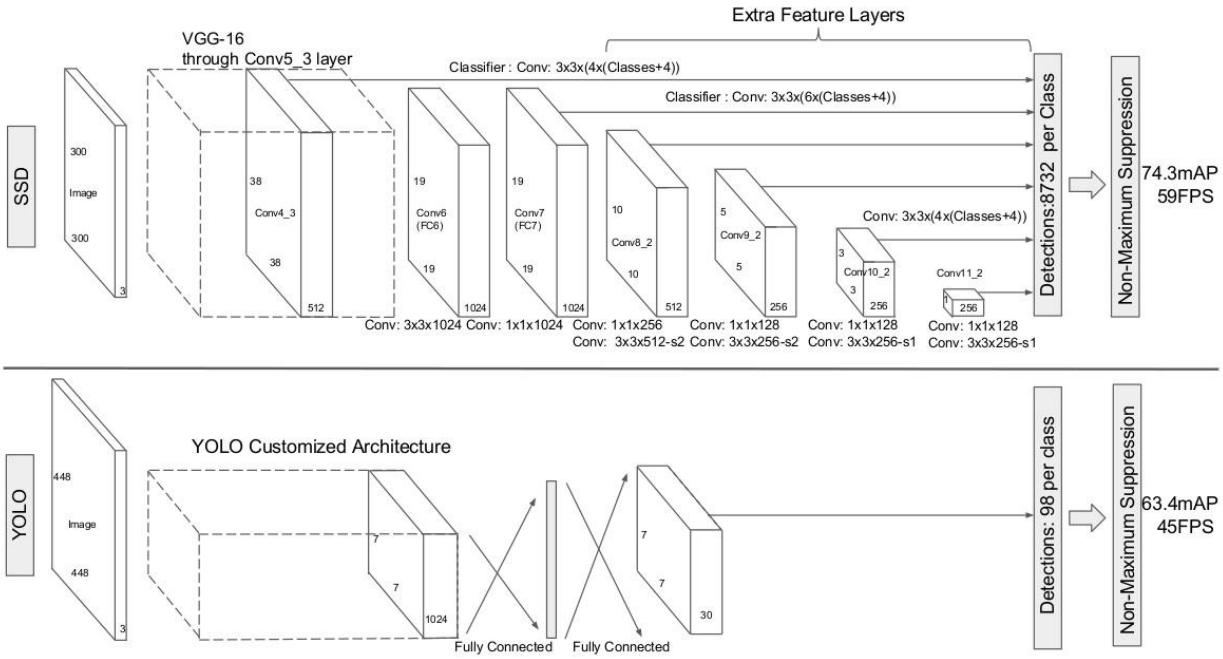
这里的smin为0.2，表示最底层的scale大小;smax为0.9，表示最高层的scale大小

2) aspect ratio计算：

We impose different aspect ratios for the default boxes, and denote them as $a_r \in \{1, 2, 3, \frac{1}{2}, \frac{1}{3}\}$. We can compute the width ($w_k^a = s_k\sqrt{a_r}$) and height ($h_k^a = s_k/\sqrt{a_r}$) for each default box. For the aspect ratio of 1, we also add a default box whose scale is $s'_k = \sqrt{s_k s_{k+1}}$, resulting in 6 default boxes per feature map location. We set the center of each default box to $(\frac{i+0.5}{|f_k|}, \frac{j+0.5}{|f_k|})$, where $|f_k|$ is the size of the k-th square feature map, $i, j \in [0, |f_k|)$. In practice, one can also design a distribution of default boxes to best fit a specific dataset. How to design the optimal tiling is an open question as well.

因此，对于每个feature map cell而言，一共有6中default box，可以看出这种default box在不同的feature层有不同的scale，在同一个feature层又有不同的aspect ratio，基本上可以覆盖输出图像中各种形状和大小的object！有个这种方式产生的负样本比正样本多得多，作者将负样本按照confidence loss进行排序，然后选择排名靠前的一些负样本作为训练，使得正负样本比例在1：3左右。

3) SSD与YOLO算法结构图对比



YOLO算法的输入是448x448x3，输出是7x7x30，这7x7个grid cell一共预测98个bounding box。SSD算法是在原来VGG16的后面添加了几个卷积层来预测offset和confidence（相比之下YOLO算法是采用全连接层(yolo_v1)），算法的输入是300x300x3，采用conv4_3，conv7，conv8_2，conv9_2，conv10_2和conv11_2的输出来预测location和confidence。

tip: 详细讲一下SSD的结构，可以参看Caffe代码。SSD的结构为conv1_1，conv1_2，conv2_1，conv2_2，conv3_1，conv3_2，conv3_3，conv4_1，conv4_2，conv4_3，conv5_1，conv5_2，conv5_3（512），fc6：3*3*1024的卷积（原来VGG16中的fc6是全连接层，这里变成卷积层，下面的fc7层同理），fc7:1*1*1024的卷积，conv6_1，conv6_2（对应上图的conv8_2），……，conv9_1，conv9_2，loss。然后针对conv4_3（4），fc7（6），conv6_2（6），conv7_2（6），conv8_2（4），conv9_2（4）的每一个再分别采用两个3*3大小的卷积核进行卷积，这两个卷积核是并列的（括号里的数字代表**default** box的数量，可以参考Caffe代码，所以上图中SSD结构的倒数第二列的数字8732表示的是所有**default** box的数量，是这么来的38*38*4+19*19*6+10*10*6+5*5*6+3*3*4+1*1*4=8732），这两个3*3的卷积核一个是用来做localization的（回归用，如果**default** box是6个，那么就有6*4=24个这样的卷积核，卷积后map的大小和卷积前一样，因为pad=1，下同），另一个是用来做confidence的（分类用，如果**default** box是6个，VOC的object类别有20个，那么就有6*（20+1）=126个这样的卷积核）。如下图是conv6_2的localizaiton的3*3卷积核操作，卷积核个数是24（6*4=24，由于pad=1，所以卷积结果的map大小不变，下同）：这里的permute层就是交换的作用，比如你卷积后的维度是32*24*19*19，那么经过交换层后就变成32*19*19*24，顺序变了而已。而flatten层的作用就是将32*19*19*24变成32*8664，32是batchsize的大小。