

## Mask RCNN - 原理解析 (一)

<https://blog.csdn.net/ghw15221836342/article/details/80084861>

Mask R-CNN 是一个**两阶段的框架**:

- 第一个阶段扫描图像并生成提议 (proposals, 即有可能包含一个目标的区域) ,
- 第二阶段分类提议并生成边界框和掩码。

Mask R-CNN 扩展自 Faster R-CNN, 由同一作者提出。

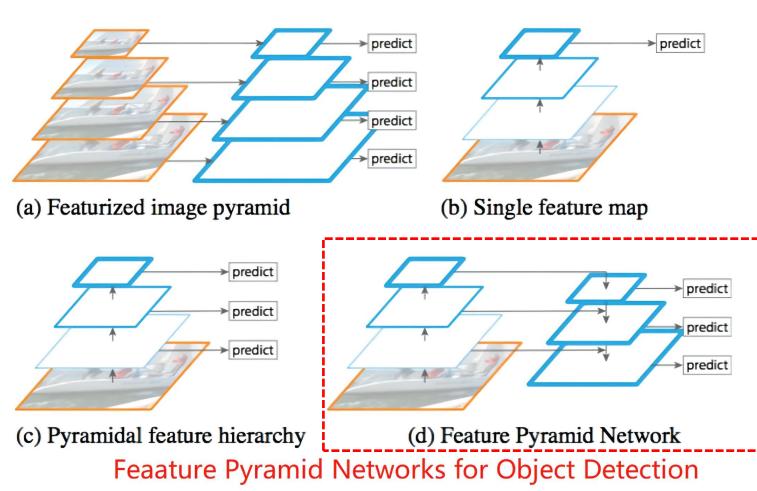
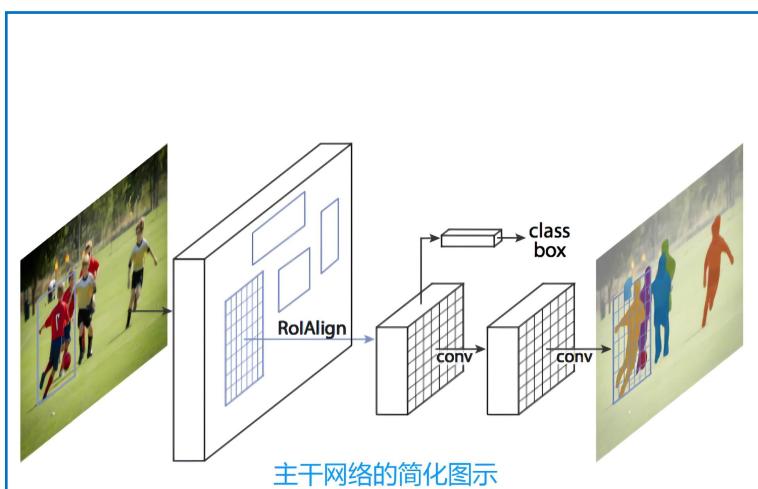
Faster R-CNN 是一个流行的目标检测框架, Mask R-CNN 将其扩展为**实例分割框架**。

Mask R-CNN 的主要构建模块:

- 主干架构
- 区域建议网络 (RPN)
- ROI 分类器和边界框回归器
- 分割掩码

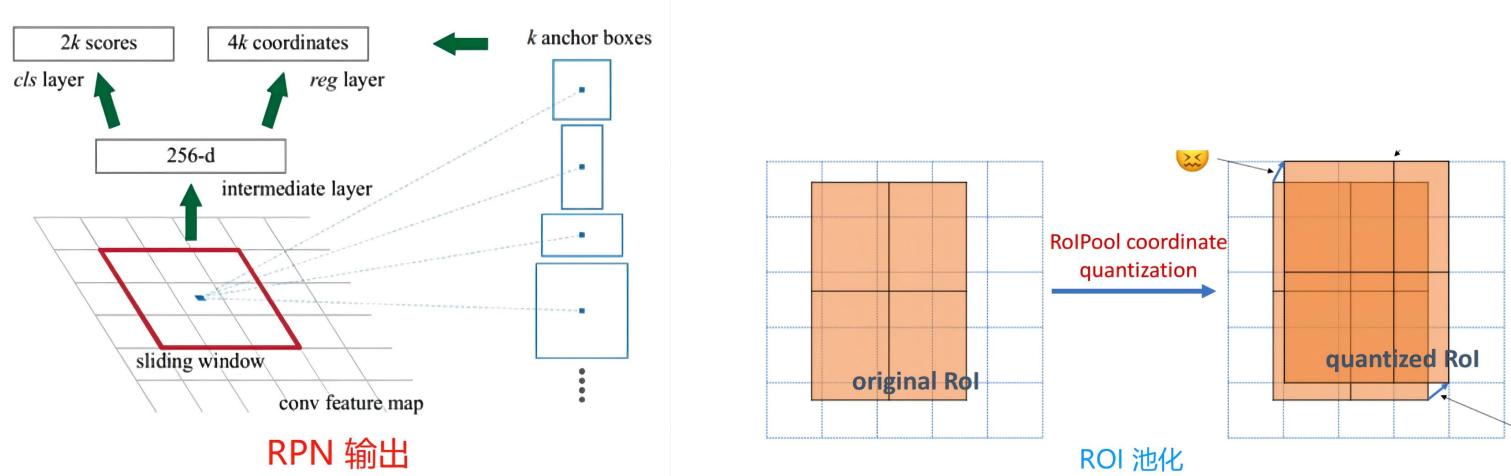
## 主干架构

- 一个**标准的卷积神经网络** (通常来说是 ResNet50 和 ResNet101), 作为**特征提取器**。
  - 底层检测的是低级特征 (边缘和角等), 较高层检测的是更高级的特征 (汽车、人、天空等)。
  - 经过主干网络的前向传播, 图像从 1024x1024x3 (RGB) 的张量被转换成 32x32x2048 的特征图。
- **特征金字塔网络 (FPN)** :
  - 作者引入的特征金字塔网络 (FPN) 是对该主干网络的扩展, 可以在多个尺度上更好地表征目标。
  - FPN 通过添加第二个金字塔提升了标准特征提取金字塔的性能, 第二个金字塔可从第一个金字塔选择高级特征并传递到底层上。通过这个过程, 它允许每一级的特征都可以和高级、低级特征互相结合。
- Mask R-CNN 实现中使用的是 **ResNet101+FPN 主干网络**。



## 区域建议网络 (RPN)

- RPN 是一个轻量的神经网络，它用滑动窗口来扫描图像，并寻找存在目标的区域。
- RPN 扫描的区域被称为 anchor，这是在图像区域上分布的矩形。实际上，在不同的尺寸和长宽比下，图像上会有将近 20 万个 anchor，并且它们互相重叠以尽可能地覆盖图像。
- RPN 扫描这些 anchor 的速度非常快。滑动窗口是由 RPN 的卷积过程实现的，可以使用 GPU 并行地扫描所有区域。
- RPN 并不会直接扫描图像，而是扫描主干特征图。这使得 RPN 可以有效地复用提取的特征，并避免重复计算。通过这些优化手段，RPN 可以在 10ms 内完成扫描（根据引入 RPN 的 Faster R-CNN 论文中所述）。在 Mask R-CNN 中，我们通常使用的是更高分辨率的图像以及更多的 anchor，因此扫描过程可能会更久。
- RPN 为每个 anchor 生成两个输出：
  - anchor 类别：前景或背景 (FG/BG)。前景类别意味着可能存在一个目标在 anchor box 中。
  - 边框精调：前景 anchor (或称正 anchor) 可能并没有完美地位于目标的中心。因此，RPN 评估了 delta 输出 (x、y、宽、高的变化百分数) 以精调 anchor box 来更好地拟合目标。
- 使用 RPN 的预测，我们可以选出最好地包含了目标的 anchor，并对其位置和尺寸进行精调。如果有多个 anchor 互相重叠，我们将保留拥有最高前景分数的 anchor，并舍弃余下的（非极大值抑制）。然后我们就得到了最终的区域建议，并将其传递到下一个阶段。

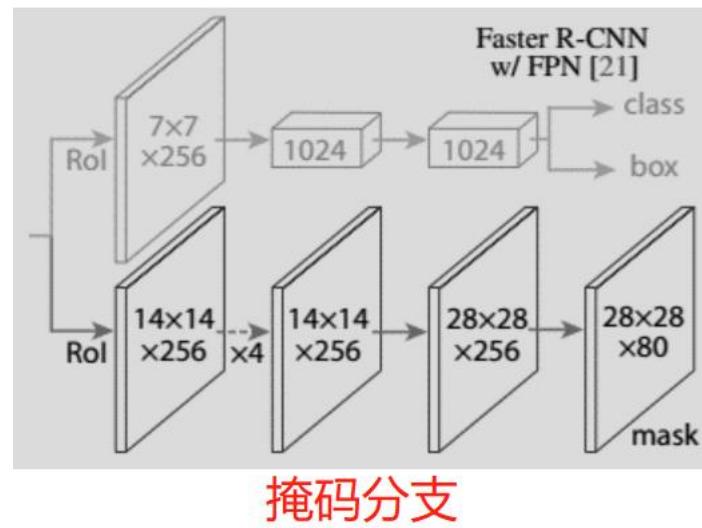


## ROI 分类器 和 边界框回归器

- 这个阶段是在由 RPN 提出的 ROI 上运行的。正如 RPN 一样，它为每个 ROI 生成了两个输出：
  - **类别：**ROI 中的目标的类别。和 RPN 不同（两个类别，前景或背景），这个网络更深并且可以将区域分类为具体的类别（人、车、椅子等）。它还可以生成一个背景类别，然后就可以弃用 ROI 了。
  - **边框精调：**和 RPN 的原理类似，它的目标是进一步精调边框的位置和尺寸以将目标封装（框住）。
- **ROI 池化**
  - 在进一步继续之前，需要先解决一些问题。分类器并不能很好地处理多种输入尺寸。它们通常只能处理固定的输入尺寸。但是，由于 RPN 中的边框精调步骤，ROI 框可以有不同的尺寸。因此，我们需要用 ROI 池化来解决这个问题。
  - ROI 池化是指裁剪出特征图的一部分，然后将其重新调整为固定的尺寸。这个过程实际上和裁剪图片并将其缩放是相似的（在实现细节上有所不同）。
- **ROIAlign**
  - Mask R-CNN 的作者提出了一种方法 **ROIAlign**，在特征图的不同点采样，并应用**双线性插值**。在我们的实现中，为简单起见，我们使用 TensorFlow 的 `crop_and_resize` 函数来实现这个过程。

## 分割掩码

- 到步骤 3 为止，我们得到的正是一个用于目标检测的 Faster R-CNN。而分割掩码网络正是 Mask R-CNN 的论文引入的附加网络。
- 掩码分支是一个卷积网络，取 ROI 分类器选择的正区域为输入，并生成它们的掩码。
- 生成的掩码是低分辨率的：**28x28 像素**。但它们是由**浮点数**表示的软掩码，相对于**二进制掩码**有**更多的细节**。
- 掩码的小尺寸属性有助于保持掩码分支网络的轻量性。
- 在训练过程中**，我们将真实的掩码缩小为 28x28 来计算损失函数，**在推断过程中**，我们将预测的掩码放大为 ROI 边框的尺寸以给出最终的掩码结果，每个目标有一个掩码。



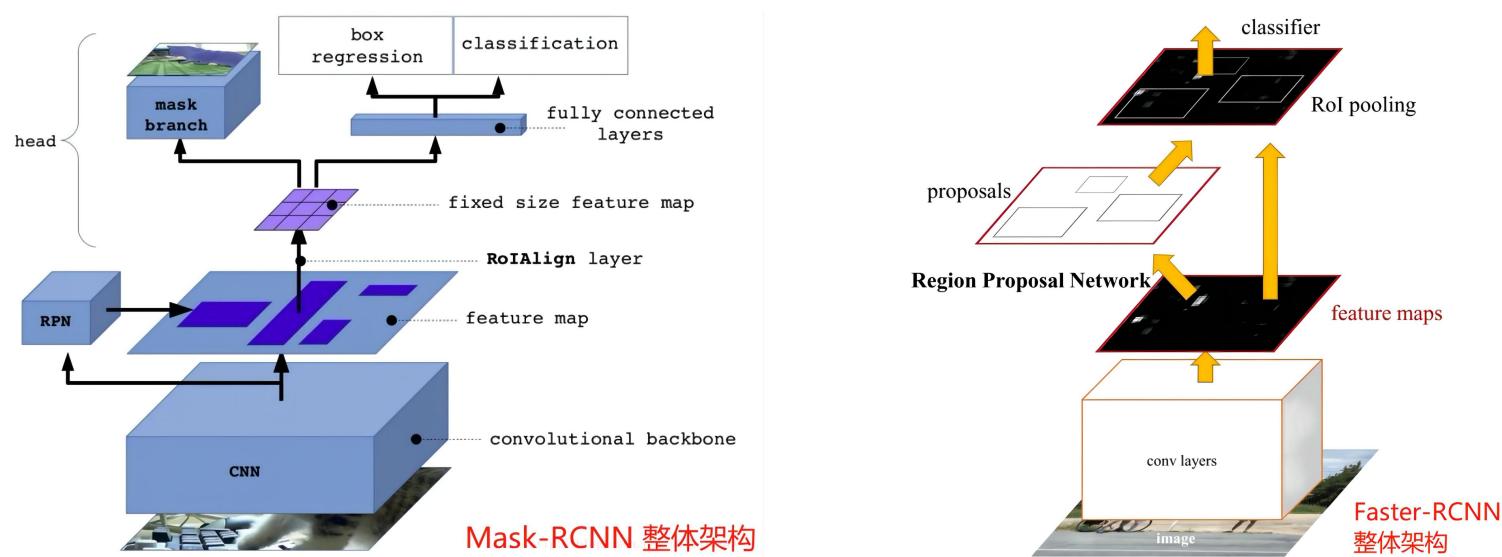
# Mask-RCNN 算法及其实现详解

<https://blog.csdn.net/remanented/article/details/79564045>

Mask-RCNN 集成了物体检测和实例分割两大功能，并且在性能上也超过了Faster-RCNN。

在Faster-RCNN的基础之上，Mask-RCNN加入了 Mask branch (FCN) 用于生成物体的掩模(object mask)，同时把RoI pooling 修改成为了RoI Align 用于处理mask与原图中物体不对齐的问题。

遵循自下而上的原则，依次的从**backbone**, **FPN**, **RPN**, **anchors**, **RoIAlign**, **classification**, **box regression**, **mask** 这几个方面讲解。

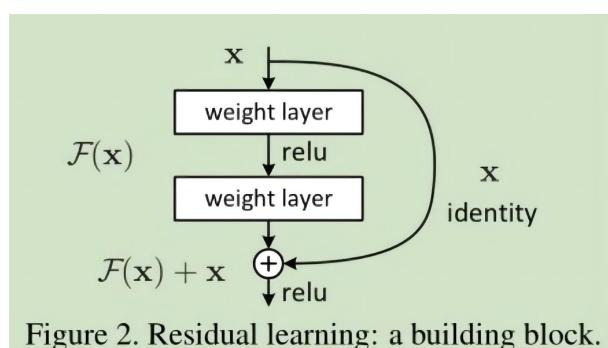


## backbone

backbone是一系列的卷积层用于提取图像的feature maps，比如可以是VGG16, VGG19, GoLeNet, ResNet50, ResNet101等，这里主要讲解的是ResNet101的结构。

ResNet (深度残差网络) 实际上就是为了能够训练更加深层的网络提供了有利的思路，毕竟之前一段时间里面一直相信深度学习中网络越深得到的效果会更加的好，但是在构建了太深层之后又会使得网络退化。ResNet使用了跨层连接，使得训练更加容易。

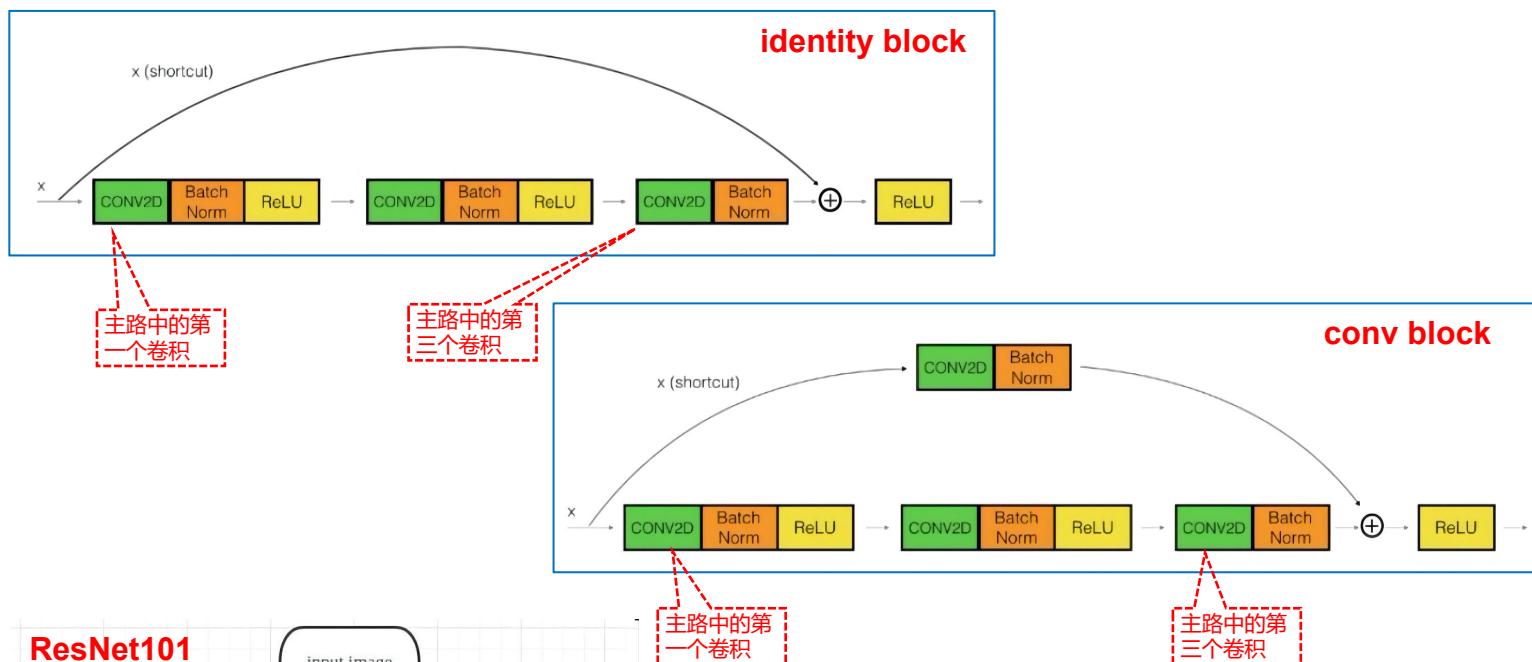
网络试图让一个block的输出为 $f(x) + x$ ，其中的 $f(x)$ 为残差，当网络特别深的时候残差 $f(x)$ 会趋近于0，从而 $f(x) + x$ 就等于 $x$ ，即实现了恒等变换，不管训练多深性能起码不会变差。



ResNet 网络中只存在两种类型的block，是这两种block在交替或者循环的使用：

- **identity block**: 该block中直接把开端的x接入到第三个卷积层的输出。注意主路上第三个卷积层使用的激活层，在相加之后才进行了ReLU的激活。
- **conv block**: 与identity block其实是差不多的，只是在shortcut上加了一个卷积层再进行相加。注意主路上的第三个卷积层和shortcut上的卷积层都没激活，而是先相加再进行激活的。

在作者的代码中，**主路中的第一个和第三个卷积都是  $1 \times 1$  的卷积**（改变的只有 feature maps 的通道大小，不改变长和宽），为了降维从而实现卷积运算的加速；注意需要保持 shortcut 和主路最后一个卷积层的 channel 要相同才能够进行相加。



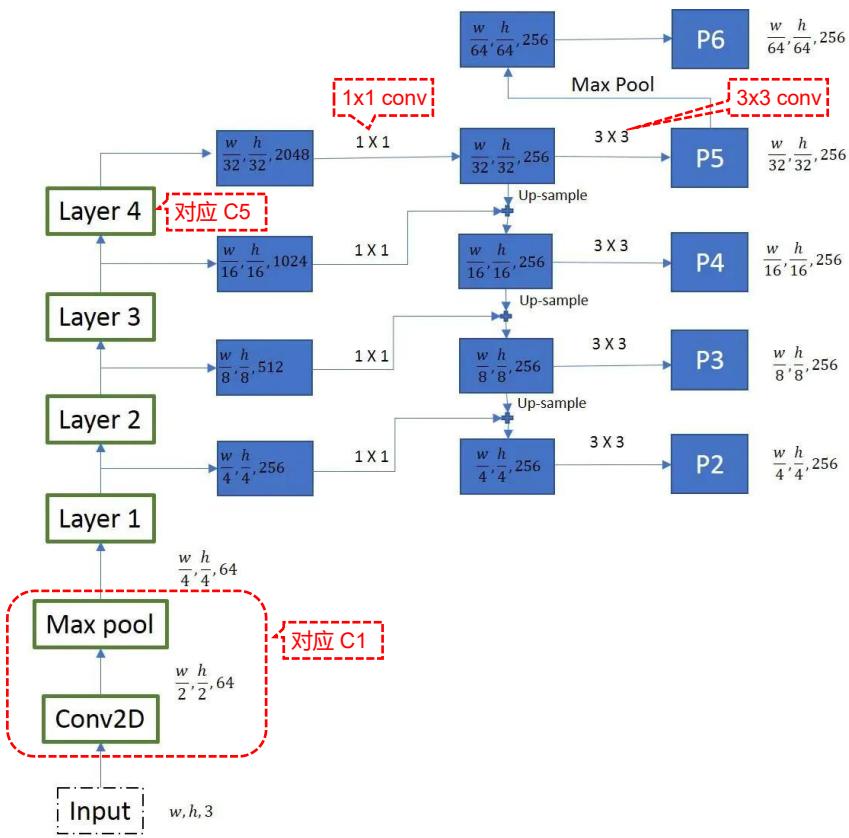
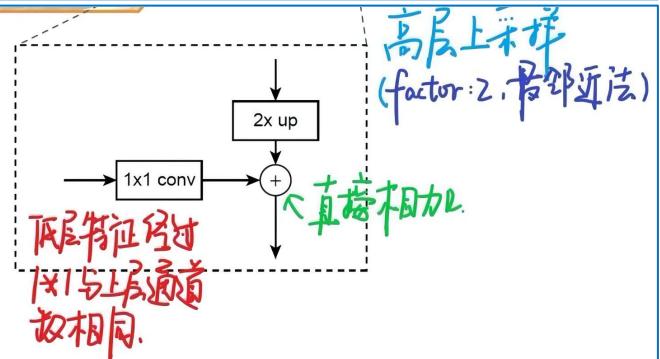
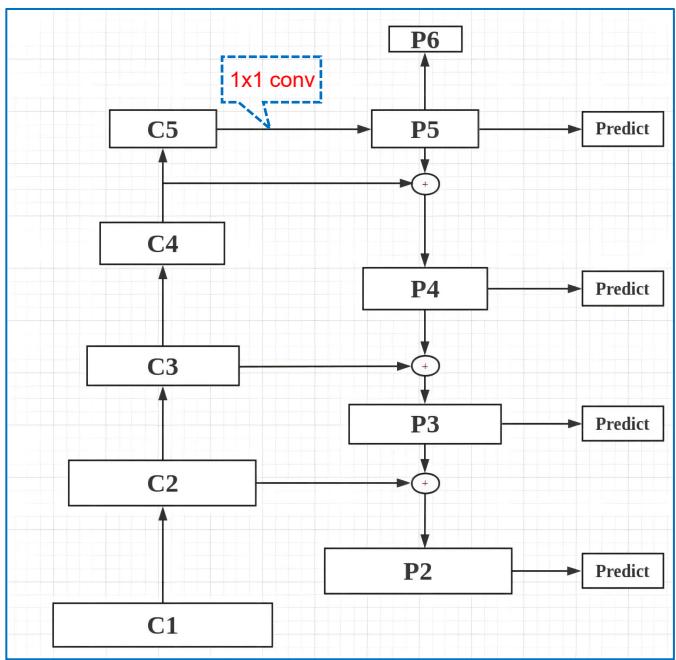
从图中可以得知ResNet分为了5个stage，C1-C5分别为每个Stage的输出，这些输出在后面的FPN中会使用到。

我们可以数数，看看是不是总共101层，数的时候除去BatchNorm层。

注：stage4 中是由一个 conv\_block 和 22 个 identity\_block，如果要改成 ResNet50 网络的话只需要调整为5个 identity\_block。

# FPN (Feature Pyramid Network)

FPN的提出是为了实现更好的feature maps融合。一般的网络都是直接使用最后一层的feature maps，虽然最后一层的feature maps语义强，但是位置和分辨率都比较低，容易检测不到比较小的物体。FPN的功能就是融合了底层到高层的feature maps，从而充分的利用了提取到的各个阶段的特征（ResNet中的C2 - C5）。



右图来源：  
目标检测系列 Mask R-CNN—FPN： <https://www.jianshu.com/p/4c933edd017f>

# Note that P6 is used in RPN, but not in the classifier heads.  
rpn\_feature\_maps = [P2, P3, P4, P5, P6]  
mrcnn\_feature\_maps = [P2, P3, P4, P5]

从上图左边中可以看出 + 的意义为：左边的底层特征层通过  $1 \times 1$  的卷积得到与上一层特征层相同的通道数；上层的特征层通过上采样得到与下一层特征层一样的长和宽再进行相加，从而得到了一个融合好的新的特征层。

举个例子说就是：C4层经过  $1 \times 1$  卷积得到与P5相同的通道，P5经过上采样后得到与C4相同的长和宽，最终两者进行相加，得到了融合层P4，其他的以此类推。

**注：**P2 - P5 是将来用于预测物体的bbox, box-regression, mask的，而 P2 - P6 是用于训练RPN的，即P6只用于RPN网络中。

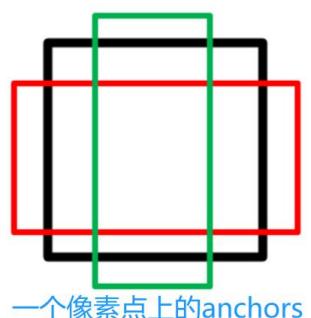
## anchors

anchors 英文翻译为锚点、锚框。是用于在feature maps的像素点上产生一系列的框，各个框的大小由 scale 和 ratio 这两个参数来确定的，比如  $\text{scale} = [128]$ ,  $\text{ratio} = [0.5, 1, 1.5]$ ，则每个像素点可以产生3个不同大小的框。这三个框是由保持框的面积不变，来通过ratio的值来改变其长宽比，从而产生不同大小的框。 $(\text{ratio} = \text{width}/\text{height})$

假设我们现在绘制feature maps上一个像素点的anchors，则能得到右图：

由于使用到了FPN，在论文中也有说到每层的feature map 的scale是保持不变的，只是改变每层的ratio，且越深scale的值就越小，因为越深的话feature map就越小。

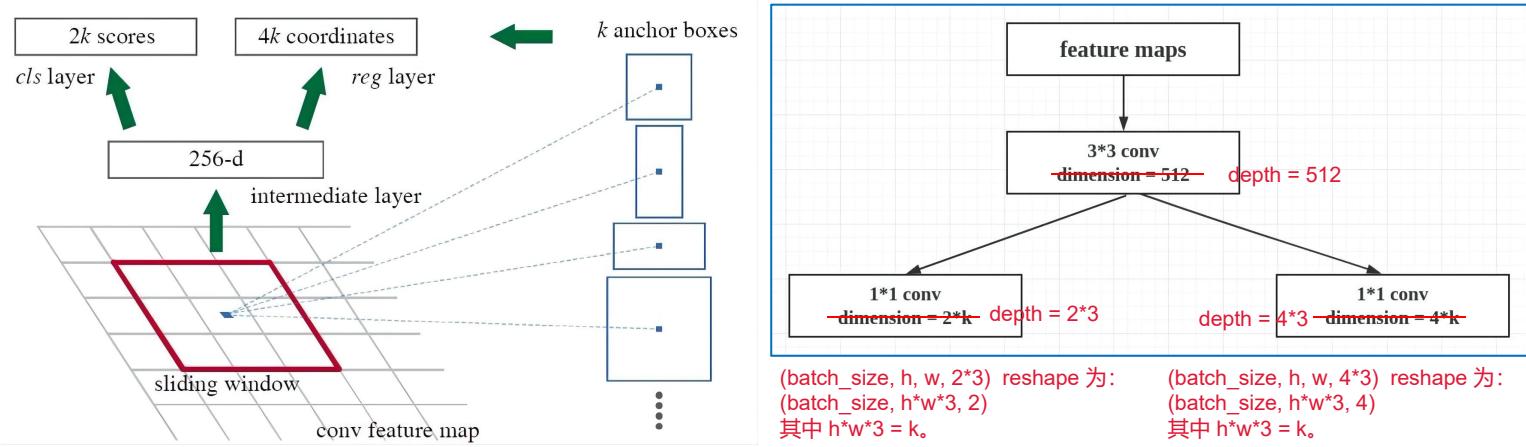
论文中提供的每层的scale为(32, 64, 128, 256, 512)，ratio为(0.5, 1, 2)，所有每一层的每一个像素点都会产生3个锚框，而总共会有15种不同大小的锚框。



# RPN (Region Proposal Network)

RNP顾名思义：区域推荐的网络，用于帮助网络推荐感兴趣的区域，也是Faster-RCNN中重要的一部分。

- **conv feature map**: 上文中的 P2-P6。
- **k anchor boxes**: 在每个 sliding window 的点上的初始化的参考区域。每个 sliding window 的点上取的 anchor boxes都一样。只要知道 sliding window 的点的坐标，就可以计算出每个 anchor box 的具体坐标。每个特征层的 anchor 数量为  $k=3 \times h \times w$ 。先确定一个base anchor，如 P6 大小为  $32 \times 32 \times 16 \times 16$ ，保持面积不变使其长宽比为 (0.5, 1, 2)，其每个像素点的位置得到3个anchors，P6 这一层产生的 anchor 数量为  $3 \times 16 \times 16$ 。
- **intermediate layer**: 作者代码中使用的是 depth 为 512，kernel (3, 3) 的conv中间层，再通过 depth 分别为  $2 \times 3$  和  $4 \times 3$ ，kernel 为  $1 \times 1$  的卷积，之后 reshape 从而获得  $2k$  scores 和  $4k$  coordinates。作者在文中解释为用全卷积方式替代全连接。
- **2k scores**: 对于每个anchor，用了softmax layer的方式，得到两个置信度。一个置信度是前景，一个置信度是背景
- **4k coordinates**: 每个窗口的 delta 坐标。这个坐标并不是anchor的绝对坐标，而是与 ground\_truth 偏差的回归，后续与 anchors 坐标进行回归。



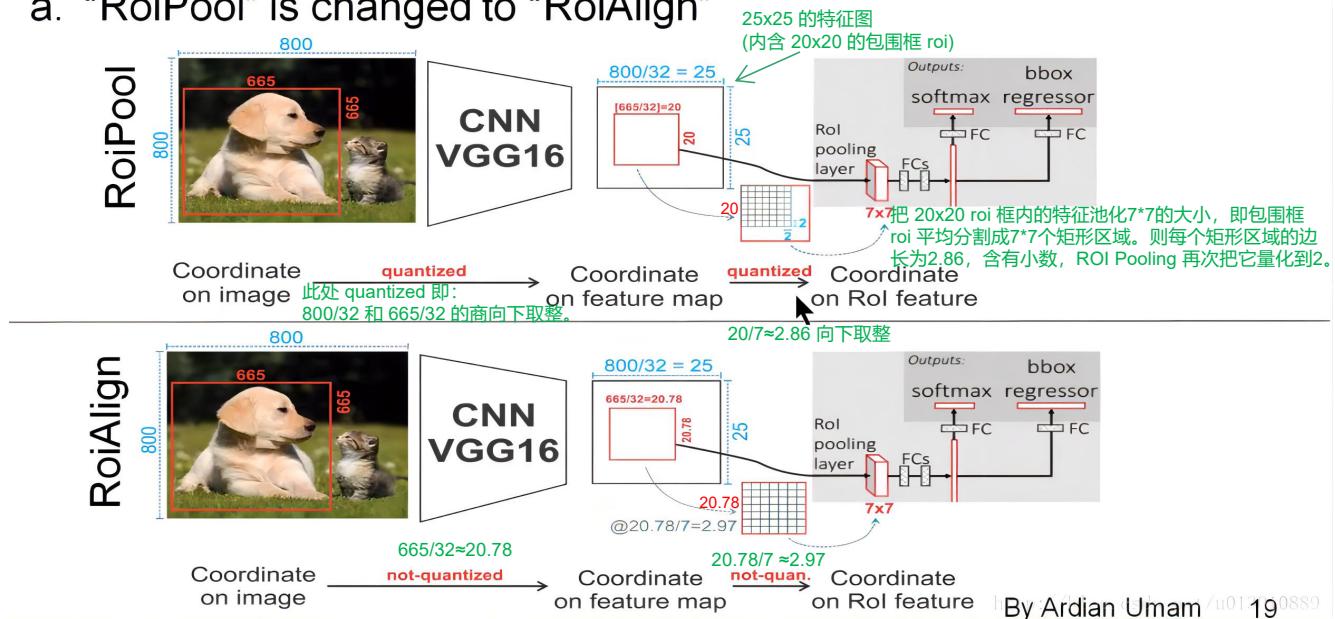
在作者代码中RPN的网络具体结构如上面右图所示。

**注：**在开始看作者代码的时候也是有些蒙圈的，为什么给RPN只传入了feature map和k值就可以，而没有给出之前创建好的anchors，后来才明白作者在数据产生那一块做了修改，他在产生数据的时候就给每一个创建好的 anchors 标注好了是 positive 还是 negative 以及需要回归的 box 值，所有只需要训练RPN就好了。

# RoIAlign (请一定要结合这篇文章来理解: <https://zhuanlan.zhihu.com/p/73113289>)

Mask-RCNN 中提出了一个新的 idea 就是 RoIAlign，其实 RoIAlign 就是在 ROI pooling 上稍微改动过来的。

## a. “RoiPool” is changed to “RoIAlign”



可以看出来在 ROI pooling 中出现了两次的取整，虽然在 feature maps 上取整看起来只是小数级别的数，但是当把 feature map 还原到原图上时就会出现很大的偏差，比如第一次的取整是舍去了 0.78 ( $665/32 \approx 20.78$ )，还原到原图时是  $0.78 * 32 = 25$ ，第一次取整就存在了 25 个像素点的偏差，在第二次的取整后的偏差更加的大。对于分类和物体检测来说可能这不是一个很大的误差，但是对于实例分割而言，这是一个非常大的偏差，因为 mask 出现没对齐的话在视觉上是很明显的。而 RoIAlign 的提出就是为了解决这个问题，解决不对齐的问题。（经过这两次量化，候选区域已经出现了较明显的偏差更重要的是，该层特征图上 0.1 个像素的偏差，缩放到原图就是 3.2 个像素。）

RoIAlign 的思想其实很简单，就是取消了取整的这种粗暴做法，而是通过**双线性插值**（听说好像有一篇论文用到了积分，而且性能得到了一定的提高）来得到固定四个点坐标像素值，从而使得不连续的操作变得连续起来，返回到原图的时候误差也就更加的小。（取消量化操作，使用双线性内插的方法获得坐标为浮点数的像素点上的图像数值，从而将整个特征聚集过程转化为一个连续的操作。）

1. 划分  $7 \times 7$  的 bin。（可以直接精确的映射到 feature map 上来划分 bin，不用第一次 ROI 的量化。即保持边界的浮点数坐标不做量化。）
2. 接着是对每一个 bin 中进行双线性插值，得到四个点。（在论文中也说到过插值一个点的效果其实和四个点的效果是一样的，在代码中作者为了方便也就采用了插值一个点）
3. 通过插完值之后再进行 max pooling 得到最终的  $7 \times 7$  的 ROI，即完成了 RoIAlign 的过程。

12	2	2	7	6	3	5	8	5	5
5	2	7	8	7	7	3	4	6	3
5	7	5	9	2	5	29	2	3	5
4	9	9	9	27	9	0	1	9	5
9	2	3	9	9	3	9	2	9	6
9	3	5	2	3	78	2	5	84	9
8	5	1	12	5	2	9	9	4	6
3	1	5	64	1	5	5	2	6	3
7	2	3	2	2	3	7	5	1	7
3	5	7	5	7	5	7	3	2	7

ROI (红色框) 分割  $7 \times 7$  的 bin

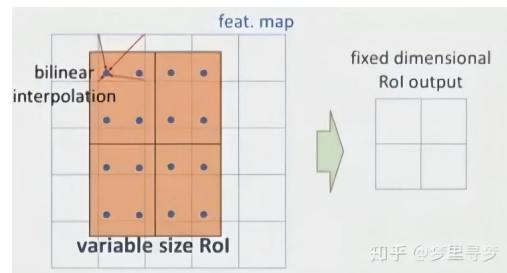
12	2	2	7	6	3	5	8	5	5
5	2	7	8	7	7	3	4	6	3
5	7	5	9	2	5	29	2	3	5
4	9	9	9	27	9	0	1	9	5
9	2	3	9	9	3	9	2	9	6
9	3	5	2	3	78	2	5	84	9
8	5	1	12	5	2	9	9	4	6
3	1	5	64	1	5	5	2	6	3
7	2	3	2	2	3	7	5	1	7
3	5	7	5	7	5	7	3	2	7

插值示意图 (一个 bin 中的四个蓝色点都是插值得出的)

为了解决ROI Pooling的上述缺点，作者提出了ROI Align这一改进的方法。

ROI Align 的思路很简单：取消量化操作，使用双线性内插的方法获得坐标为浮点数的像素点上的图像数值，从而将整个特征聚集过程转化为一个连续的操作。值得注意的是，在具体的算法操作上，ROI Align并不是简单地补充出候选区域边界上的坐标点，然后将这些坐标点进行池化，而是重新设计了一套比较优雅的流程，如下图所示：

- 遍历每一个候选区域，保持浮点数边界不做量化。
- 将候选区域分割成 $k \times k$ 个单元，每个单元的边界也不做量化。
- 在每个单元中计算固定四个坐标位置，用双线性内插的方法计算出这四个位置的值，然后进行最大池化操作。



这里对上述步骤的第三点作一些说明：

这个固定位置是指在每一个矩形单元（bin）中按照固定规则确定的位置。

- 比如，如果采样点数是1，那么就是这个单元的中心点。
- 如果采样点数是4，那么就是把这个单元平均分割成四个小方块以后它们分别的中心点。

显然这些采样点的坐标通常是浮点数，所以需要使用插值的方法得到它的像素值。

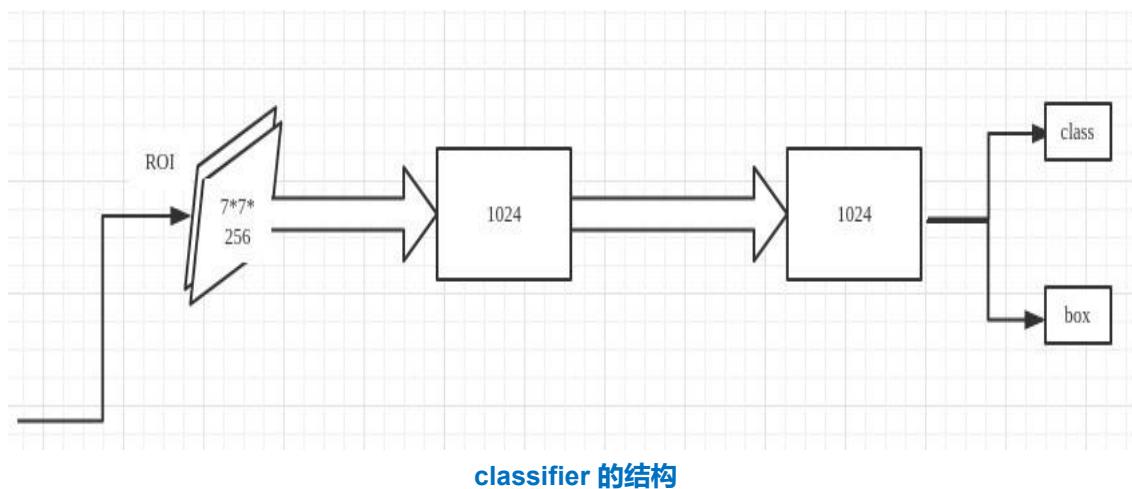
在相关实验中，作者发现将采样点设为4会获得最佳性能，甚至直接设为1在性能上也相差无几。事实上，ROI Align 在遍历取样点的数量上没有 ROI Pooling 那么多，但却可以获得更好的性能，这主要归功于解决了 misalignment 的问题。

值得一提的是，我在实验时发现，ROI Align 在 VOC2007 数据集上的提升效果并不如在 COCO 上明显。经过分析，造成这种区别的原因是 COCO 上小目标的数量更多，而小目标受 misalignment 问题的影响更大（比如，同样是0.5个像素点的偏差，对于较大的目标而言显得微不足道，但是对于小目标，误差的影响就要高很多）。

## classifier

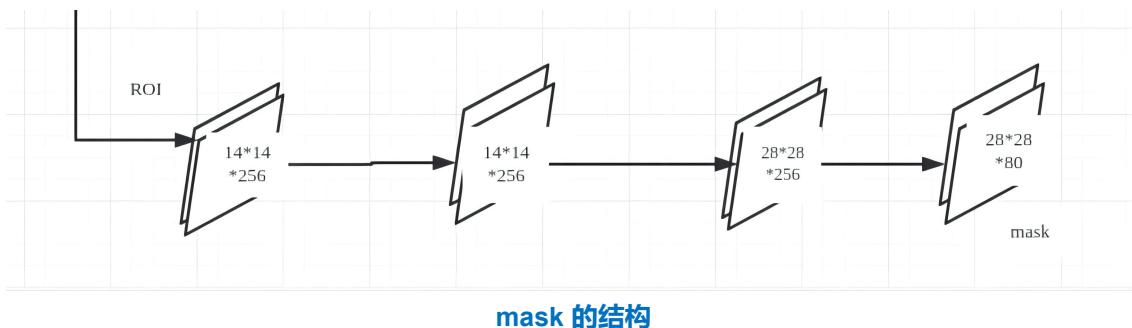
其中包括了物体检测最终的 classes 和 bounding boxes。该部分是利用了之前检测到的 ROI 进行分类和回归（是分别对每一个 ROI 进行）。

论文中提到用1024个神经元的全连接网络，但是在代码中作者用卷积深度为1024的卷积层来代替这个全连接层。



## mask

mask 的预测也是在 ROI 之后的，通过FCN (Fully Convolution Network) 来进行的。注意这个是实现的语义分割而不是实例分割。因为每个ROI只对应一个物体，只需对其进行语义分割就好，相当于实例分割了，这也是Mask-RCNN与其他分割框架的不同，是先分类再分割。



对于每一个 ROI 的mask都有80类，因为 coco 上的数据集是80个类别，并且这样做的话是为了减弱类别间的竞争，从而得到更加好的结果。

该模型的训练和预测是分开的，不是套用同一个流程。

- 在训练的时候，classifier 和 mask 都是同时进行的；
- 在预测的时候，显示得到 classifier 的结果，然后再把此结果传入到 mask 预测中得到 mask，有一定的先后顺序。

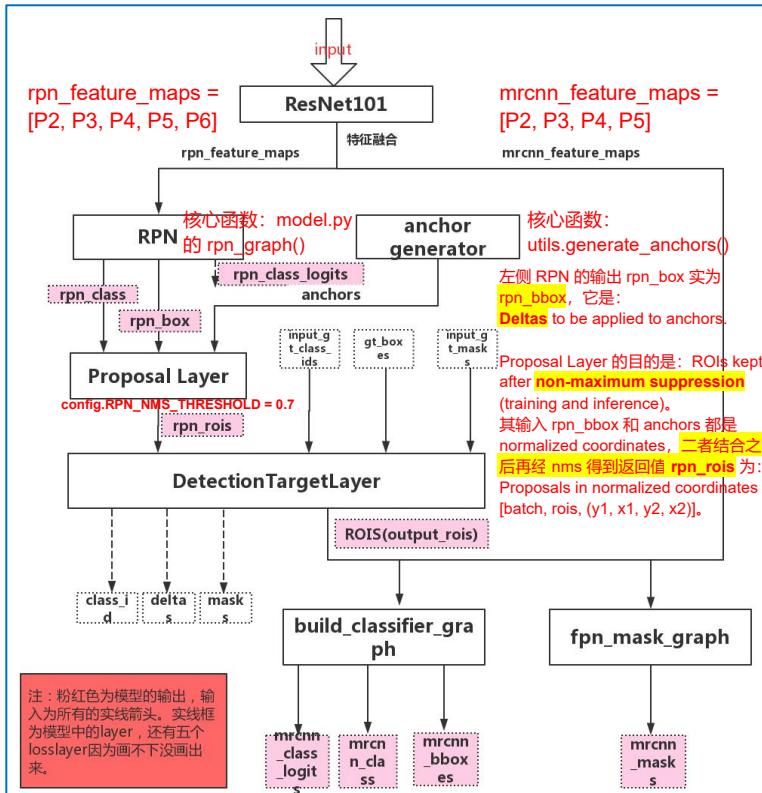
# Mask-RCNN 代码实现

文中代码的作者是Matterport。代码github地址：[https://github.com/matterport/Mask\\_RCNN](https://github.com/matterport/Mask_RCNN)

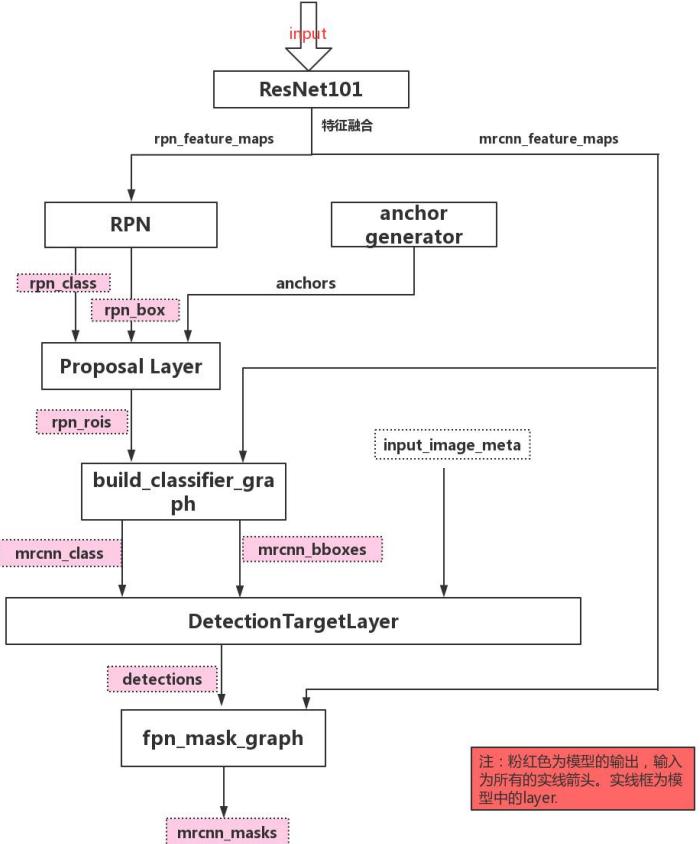
文中详细的介绍了各个部分，以及给了demo和各个实现的步骤及其可视化。

## 代码总体框架

先贴出我（指这篇文章作者）对作者（指github代码的作者）代码流程的理解，及其画出的流程图。



代码中 training 的流程图



代码中 predict 的流程图

可以清晰的看出training和predict过程是存在较大的差异的，也是之前说过的，training 的时候 mask 与 classifier 是并行的，predict 时候是先classifier再 mask，并且两个模型的输入输出差异也较大。

上面两图中的

- rpn\_box 为:  $[batch, H * W * anchors\_per\_location, (dy, dx, log(dh), log(dw))]$  Deltas to be applied to anchors.
- rpn\_class 为:  $[batch, H * W * anchors\_per\_location, 2]$  Anchor classifier probabilities.
- rpn\_class\_logits 为:  $[batch, H * W * anchors\_per\_location, 2]$  Anchor classifier logits (before softmax)

接下来的代码实现部分，参照这篇文章中行文到此处特别提到的参考文献：（见下一页另起一篇笔记专门来总结该文）

**TensorFlow实战：Chapter-8上(Mask R-CNN介绍与实现)**

<https://blog.csdn.net/u011974639/article/details/78483779?locationNum=9&fps=1>

# TensorFlow实战：Chapter-8上(Mask R-CNN介绍与实现)

<https://blog.csdn.net/u011974639/article/details/78483779?locationNum=9&fps=1>

## Mask R-CNN 论文回顾

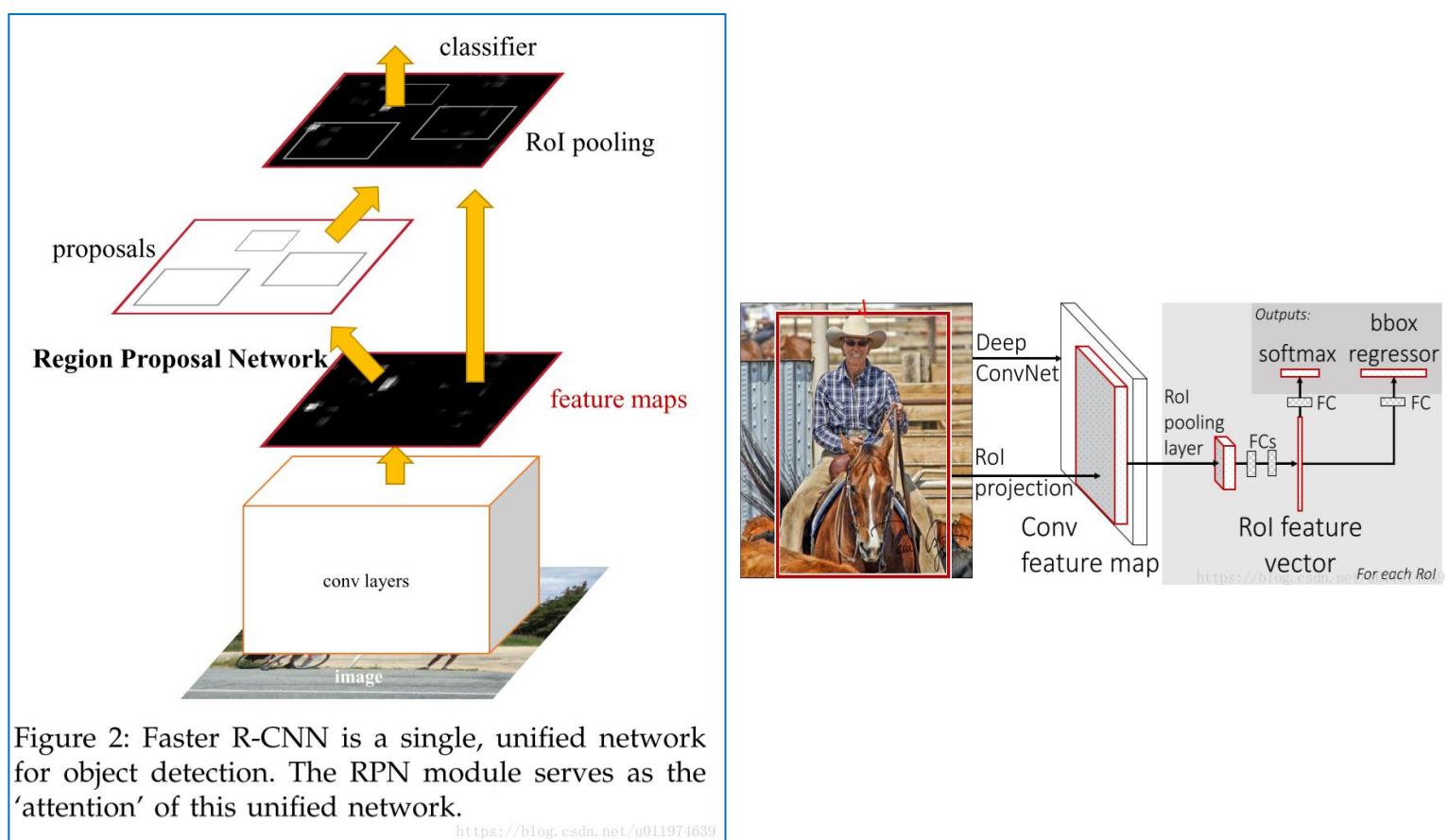
Mask R-CNN (简称 MRCNN )是基于 R-CNN 系列、FPN、FCIS 等工作之上的，MRCNN 的思路很简洁：Faster R-CNN 针对每个候选区域有两个输出：种类标签和bbox的偏移量。那么 MRCNN 就在 Faster R-CNN 的基础上通过增加一个分支进而再增加一个输出，即物体掩膜 (object mask)。

Faster R-CNN 主要由两个阶段组成：

- 区域候选网络 (Region Proposal Network, RPN)
- 基础的 Fast R-CNN 模型

RPN用于产生候选区域。 (下图左)

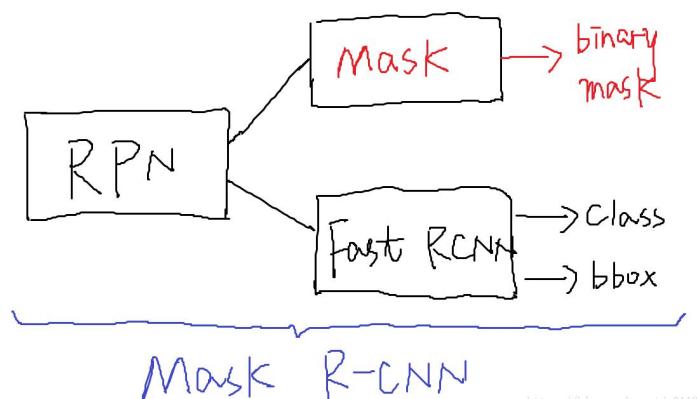
Fast R-CNN 通过 RoIPool 层对每个候选区域提取特征，从而实现目标分类和 bbox回归。 (下图右)



MRCNN 采用和 Faster R-CNN 相同的两个阶段：

- 具有相同的第一层(即RPN)
- 第二阶段，除了预测种类和bbox回归，并且并行的对每个RoI预测了对应的二值掩膜(binary mask)。

示意图如右所示：



这样做可以将整个任务简化为mulit-stage pipeline，解耦了多个子任务的关系，现阶段来看，这样做好处颇多。

<https://blog.csdn.net/u011974639>

# 损失函数的定义

依旧采用的是多任务损失函数，针对每个 RoI 定义为：

$$L = L_{cls} + L_{box} + L_{mask}$$

$L_{cls}$ ,  $L_{box}$  与 Faster R-CNN 的定义类似，这里主要看  $L_{mask}$ 。

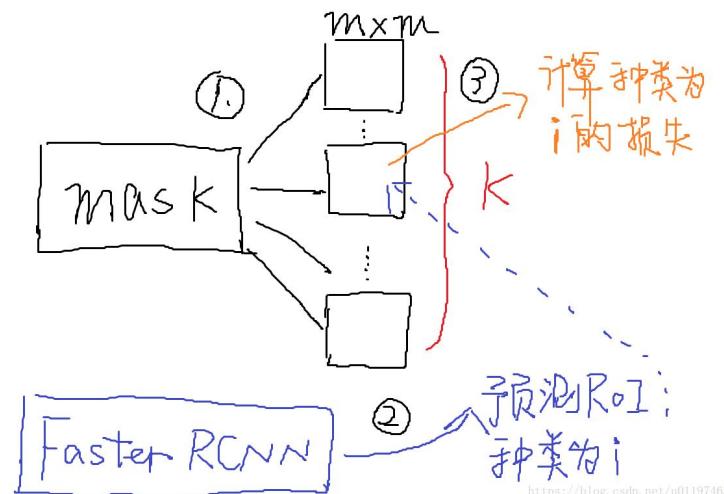
掩膜分支针对每个 RoI 产生的输出，即 **K 个分辨率相同的二值掩膜**，为分类物体的种类数目。依据预测类别分支预测的类型，只将第 i 个二值掩膜输出记为。

掩膜分支的损失计算如下示意图：

- mask branch 预测 K 个种类的  $m \times m$  二值掩膜输出。
- 依据种类预测分支(Faster R-CNN部分)预测结果：当前 RoI 的物体种类为 i。
- 第 i 个二值掩膜输出就是该RoI的损失 $L_{mask}$ 。

对于预测的二值掩膜输出，我们对每个像素点应用 sigmoid 函数，整体损失定义为 **平均二值交叉损失熵**。

引入预测个输出的机制，允许每个类都生成独立的掩膜，**避免类间竞争**。这样做解耦了掩膜和种类预测。不像 FCN 的方法，在每个像素点上应用 softmax 函数，整体采用的多任务交叉熵，这样会导致类间竞争，最终导致分割效果差。



## 掩膜表示到 RoIPool 层

在 Faster R-CNN 上预测物体标签或 bbox 偏移量是将 feature map 压缩到 FC 层最终输出 vector，压缩的过程丢失了空间上(平面结构)的信息，而掩膜是对输入目标做空间上的编码，直接用卷积形式表示像素点之间的对应关系那是最好的。

输出掩膜的操作是不需要压缩输出 vector，所以可以使用 **FCN(Full Convolutional Network)**，不仅效率高，而且参数量还少。为了更好的表示出 RoI 输入和 FCN 输出的 feature 之间的像素对应关系，提出了 **RoIPool 层**。

先回顾一下 **RoIPool 层**：

其核心思想是将不同大小的 RoI 输入到 RoIPool 层，RoIPool 层将 RoI 量化成不同粒度的特征图（量化成一个一个 bin），在此基础上使用池化操作提取特征。

下图左是 SPPNet 内对 RoI 的操作，在 Faster R-CNN 中只使用了一种粒度的特征图。平面示意图见下图右。

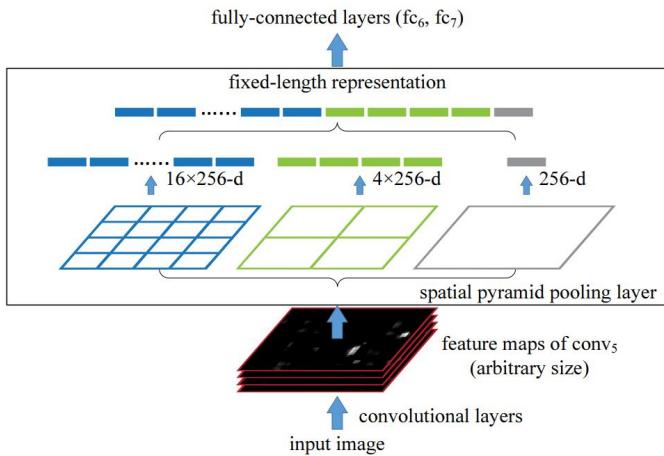
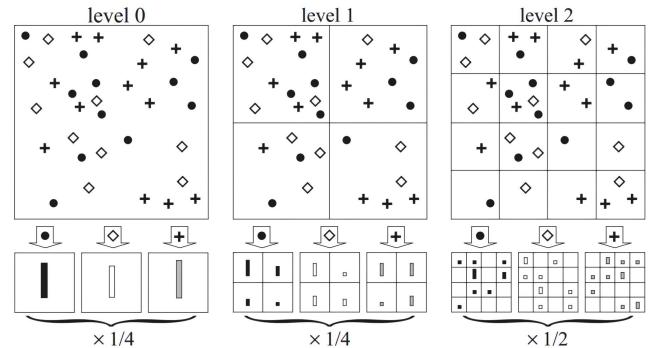
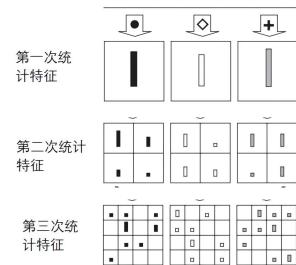


Figure 3: A network structure with a **spatial pyramid pooling layer**. Here 256 is the filter number of the conv<sub>5</sub> layer, and conv<sub>5</sub> is the last convolutional layer.

<https://blog.csdn.net/u011974639>



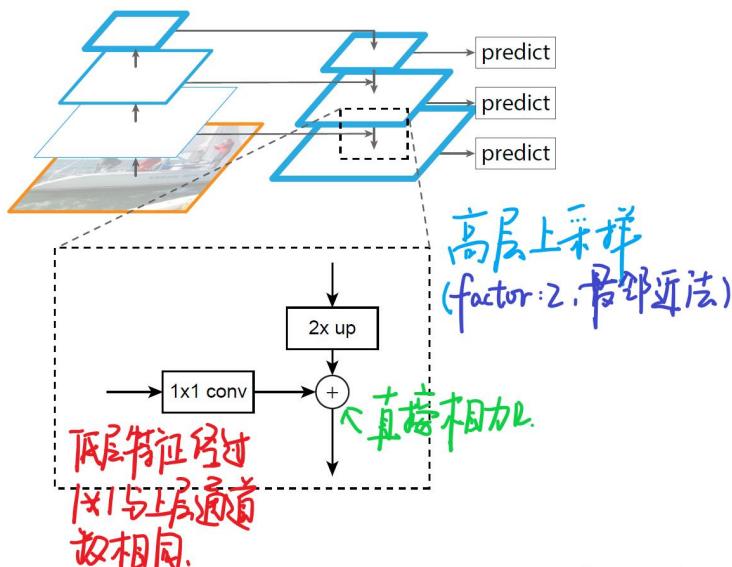
<https://blog.csdn.net/u011974639>



<https://blog.csdn.net/u011974639>

这里面存在一些问题，在上面的量操作上，实际计算中是使用的是**量化的步长**，是舍入操作(rounding)。这套量化舍入操作在提取特征时有着较好的鲁棒性(检测物体具有平移不变性等)，但是这很不利于掩膜定位，有较大负面效果。

针对这个问题，提出了 RoIAlign 层，避免了对 RoI 边界或 bin 的量化操作，在扩展 feature map 时使用双线性插值算法。这里实现的架构要看 FPN 论文。



一开始的 Faster R-CNN 是基于最上层的特征映射做分割和预测的，这会丢失高分辨下的信息，直观的影响就是丢失小目标检测，对细节部分丢失不敏感。

受到 SSD 的启发，FPN 也使用了多层特征做预测。这里使用的 top-down 的架构，是将高层的特征反卷积带到低层的特征(即有了语义，也有精度)，而在 MRCNN 论文里面说的双线性差值算法就是这里的 top-down 反卷积是用的插值算法。

## 总结：

MRCNN 有着优异的效果，除去了掩膜分支的作用，很大程度上是因为基础特征网络的增强，论文使用的是 ResNeXt101 + FPN 的 top-down 组合，有着极强的特征学习能力，并且在实验中夹杂这多种工程调优技巧。

但是 MRCNN 的缺点也很明显，需要大的计算能力并且速度慢，这离实际应用还是有很长的路。

# Region Proposal Network (RPN) —— 区域建议网络

## 参考文献:

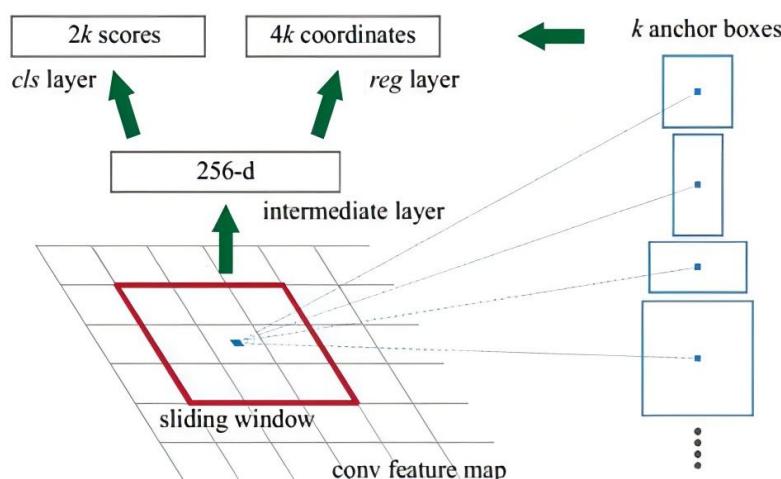
- 【论文阅读基础知识整理】区域建议网络RPN: [https://blog.csdn.net/qq\\_45603919/article/details/113862976](https://blog.csdn.net/qq_45603919/article/details/113862976)
- RPN: Region Proposal Networks (区域候选网络): <https://blog.csdn.net/zbzcDZF/article/details/88578661>
- Region Proposal Network(RPN): <https://zhuanlan.zhihu.com/p/106192020>

## RPN的输入与输出

- RPN的输入: 经过特征提取网络得到的特征图 (feature map)。
- RPN的输出: 在原图上我们要获得的是候选框, 候选框通过框中有无物体, 坐标等数据进行表示。 (输出即为这些数据)

## RPN核心: Anchor

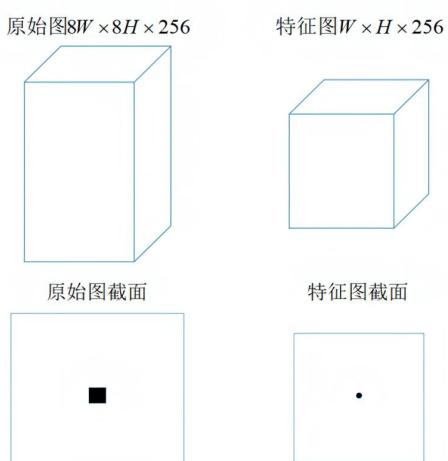
Anchor 也叫锚点, 这里有个很重要的细节: anchor 对应的是原始图而不是特征图。



这张图很清楚的展示了 RPN 的网络结构, 可以看到一开始的输入是特征图 (feature map), 最后输出为得分 (score) 和坐标 (coordinates)。

Feature map 通过 sliding window (3x3的卷积) 得到一个 $256 \times (W \times H)$ 的向量, 可以理解为有 $W \times H$ 个 $256$ 维的向量。

然后我们对每一个 $256$ 维的向量分别进行两次 $1 \times 1$ 的卷积得到 $2 \times (W \times H)$ 和 $4 \times (W \times H)$ 的特征图, 可以理解为这两个特征图中包含了 $W \times H$ 个结构, 每个结果包含了两个分数 (物体在或不在候选框的概率), 四个坐标 ( $x, y, w, h$ )。

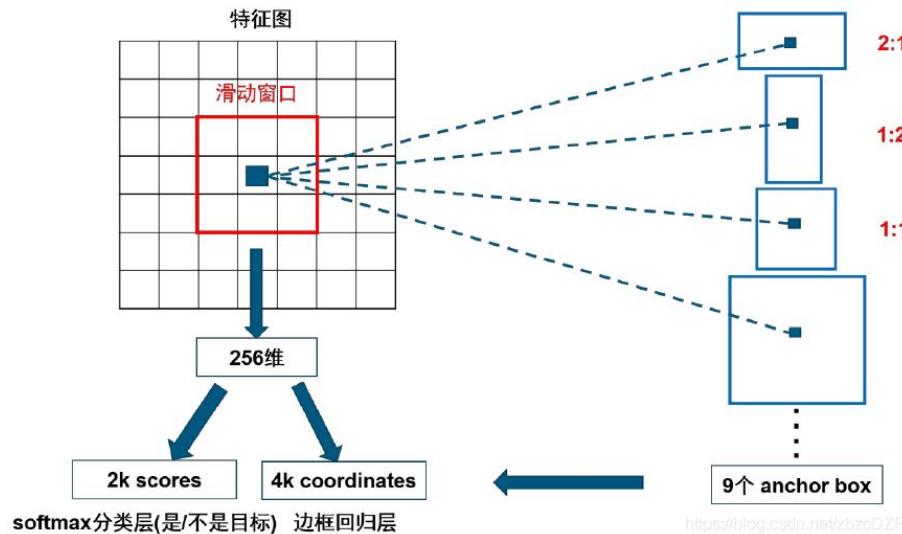


从左图看, 可以这么理解, 在特征图中的每个像素点映射到原图就是一个框。例如原始图的 $W$ 与特征图的 $W$ 比例为 $8:1$ , 则特征图中的一个像素点映射到原始图中就是一个 $8 \times 8$ 的框。然后我们将这个 $8 \times 8$ 的框的中心点或者右上角的点 (这里在代码中可以自己设置) 作为 anchor/锚点。

获得 anchor 后, 以 anchor 为中心得到 9 个基本候选框 (三种尺度, 三种比例尺), 这里的 9 也对应了 RPN 结构图中的  $k$ 。所以最后可以获得  $9 \times W \times H$  个候选框遍布整个原始图, 每个候选框又包含了 6 个信息 (框中有无物体的得分, 以及候选框的四个坐标), RPN 网络结构图中每个 $256$ 维向量可以视作一个像素点, 映射到原图得到 9 个候选框 (18 (2x9) 个分数, 36 (4x9) 个坐标)。

得到候选框后, 通过与 ground truth 的 IOU 进行正负样本的分类, 以及 NMS 等操作进行候选框的保留, 回归出准确的候选区域。

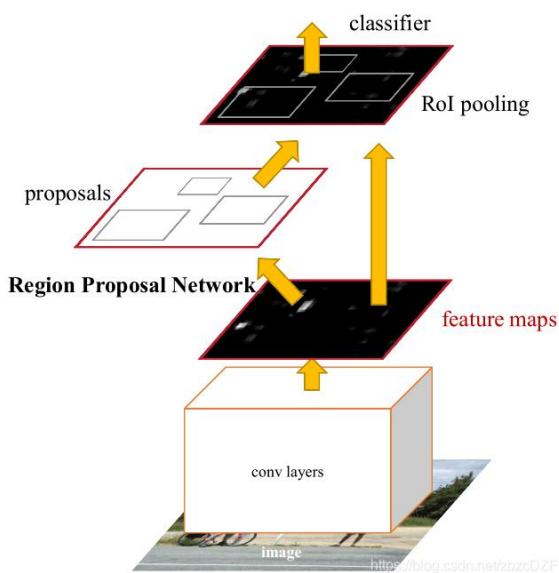
针对该像素点的每个候选框需要判断其是不是目标区域，如果是目标区域，其边框位置如何确定，具体过程如图2所示，在RPN头部，通过以下结构生成  $k$  个 anchor。



针对特征图中的某一个位置的像素点，对应会有 9 个候选框。因为输入 RPN 中有 256 个通道的特征图，所以要同时对每个通道该位置的像素点都使用不同的  $3 \times 3$  的滑动窗口进行卷积，最后将所有通道得到的该位置像素点的卷积值都加起来，得到一个新的特征值，最终使用 256 组这样的  $3 \times 3$  的卷积核，就会得到一个新的 256 维的向量，这个 256 维的向量就是用来预测该位置的像素点的，该像素点对应的 9 个候选框共享这 256 维向量。

256 维向量后面对应两条分支：

- 一条目标和背景的二分类 (classification)，通过  $1 \times 1 \times 256 \times 18$  的卷积核得到  $2k$  个分数， $k$  等于候选框的个数 9，表示这 9 个 anchor 是背景的 score 和目标的 score。
- 如果候选框是目标区域，就去判断该目标区域的候选框位置在哪，这个时候另一条分支就过  $1 \times 1 \times 256 \times 36$  的卷积核得到  $4k$  个坐标，每个框包含 4 个坐标 ( $x, y, w, h$ )，就是 9 个候选区域对应的框应该偏移的具体位置  $\Delta x_{center}, \Delta y_{center}, \Delta width, \Delta height$ 。如果候选框不是目标区域，就直接将该候选框去除掉，不再进行后续位置信息的判断操作。



Faster-RCNN 的基本结构 (来自原论文)

$$\begin{aligned} t_x &= (x - x_a)/w_a, & t_y &= (y - y_a)/h_a, \\ t_w &= \log(w/w_a), & t_h &= \log(h/h_a), \\ t_x^* &= (x^* - x_a)/w_a, & t_y^* &= (y^* - y_a)/h_a, \\ t_w^* &= \log(w^*/w_a), & t_h^* &= \log(h^*/h_a), \end{aligned}$$

## 分类分支：

考察训练集中的每张图像（含有人工标注的gt box）的所有anchor划分正负样本：

- (1) 对每个标注的 gt box 区域，与其重叠比例最大的 anchor 记为正样本，保证每个 gt 至少对应一个正样本 anchor。
- (2) 对 (1) 中剩余的 anchor，如果其与某个标注区域重叠比例大于 0.7，记为正样本（每个 gt 可能会对应多个正样本 anchor）。但每个正样本 anchor 只可能对应一个 gt；如果其与任意一个标注的重叠比例都小于 0.3，记为负样本。

## 回归分支：

- $x, y, w, h$  分别表示 box 的中心坐标和宽高， $x, x_a, x^*$  分别表示 predicted box, anchor box, and ground truth box ( $y, w, h$  同理)。
- $t_i$  表示 predict box 相对于 anchor box 的偏移，
- $t_i^*$  表示 ground true box 相对于 anchor box 的偏移，学习目标就是让前者 ( $t_i$ ) 接近后者 ( $t_i^*$ ) 的值。

在 RPN 中部，分类分支(cls) 和 边框回归分支(bbox reg) 分别对这堆 anchor 进行各种计算。在 RPN 末端，通过对两个分支的结果进行汇总，来实现对 anchor 的 **初步筛选** (先剔除越界的anchor，再根据 cls 结果通过 非极大值抑制(NMS) 算法去重) 和 **初步偏移** (根据 bbox reg结果 )，此时输出的 bbox 就都改头换面叫 Proposal 了。

偏移公式如下。an 就是 anchor 的框，pro 就是最终得出回归后的边界框，到这里我们的 proposals 就选好了：

$$\begin{aligned}x_i^{pro} &= x_i^{an} + dx_l^{reg} * w_l^{an} \\y_j^{pro} &= y_j^{an} + dy_l^{reg} * h_l^{an} \\w_l^{pro} &= w_l^{an} * e^{dw_l} \\h_l^{pro} &= h_l^{an} * e^{dh_l}\end{aligned}$$

## 非极大值抑制 (Non-maximum suppression) :

由于 anchor 一般是有重叠 (overlap)，因此，相同 object 的 proposals 也存在重叠。为了解决重叠 proposal 问题，采用 NMS 算法处理：两个 proposal 间 IoU 大于预设阈值，则丢弃 score 较低的 proposal。

IoU 阈值的预设需要谨慎处理，如果IoU值太小，可能丢失 objects 的一些 proposals；如果IoU值过大，可能会导致 objects 出现很多 proposals。IoU典型值为0.6。

## Proposal选择：

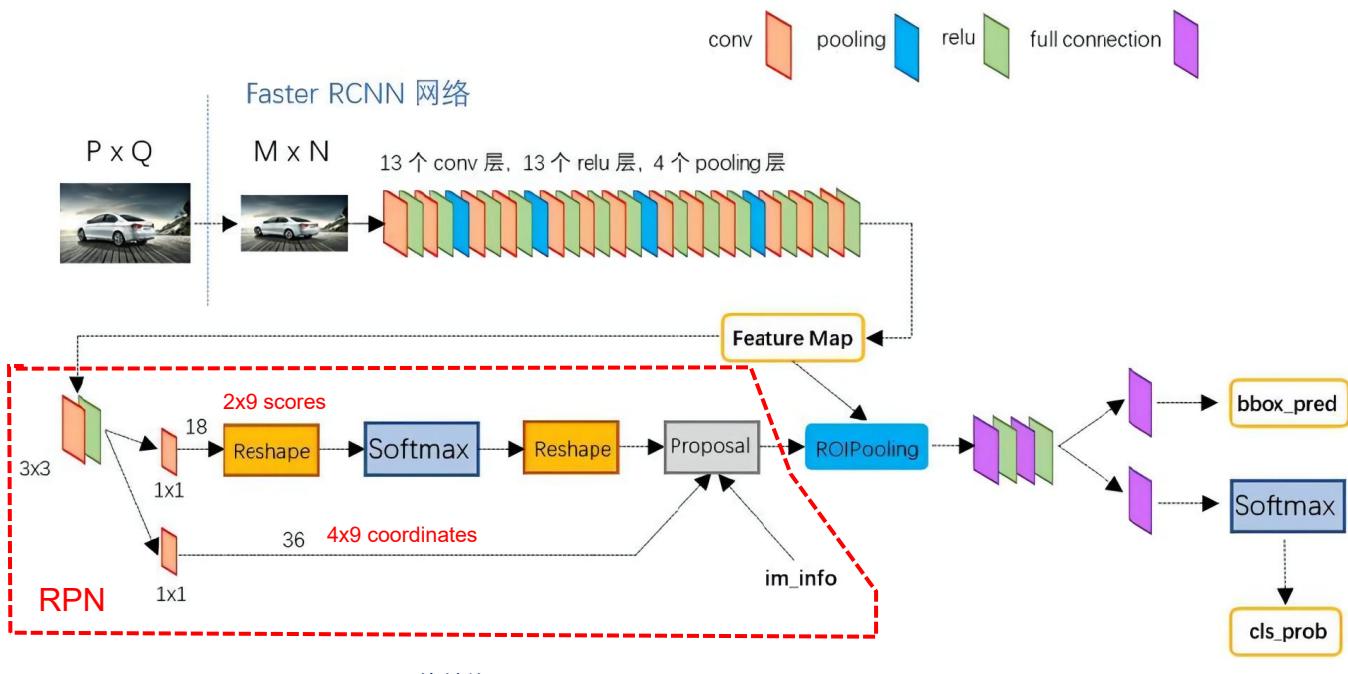
NMS 处理后，根据 score 对 top N个 proposals 排序。在 Faster R-CNN 论文中 N=2000，其值也可以小一点，如50，仍然能得到好的结果。

经典的检测方法生成检测框都非常耗时, Faster-RCNN 直接使用 RPN 生成检测框, 能极大提升检测框的生成速度。RPN (Region Proposal Network) 用于生成 候选区域(Region Proposal)。

RPN 的输入为 backbone (VGG16, ResNet, etc) 的输出 (简称 feature maps) 。

## RPN 包括以下部分:

- 生成 anchor boxes。
- 判断每个 anchor box 为 foreground(包含物体) 或者 background(背景), 二分类。
- 边界框回归(bounding box regression) 对 anchor box 进行微调, 使得 positive anchor 和真实框(Ground Truth Box)更加接近。

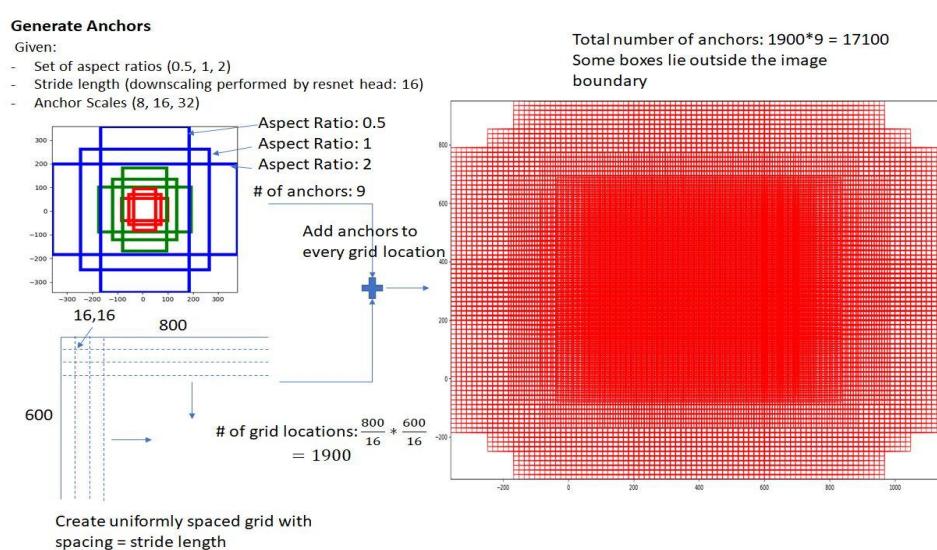


## anchor boxes

又称为『锚框』, 实际是直接运行以下代码就能产生的矩阵框: (略, 详见原网址)

遍历, 为 feature maps 的每一个点都配9个锚框, 作为初始的检测框。虽然这样得到的检测框很不准确, 但后面可通过2次 bounding box regression 来修正检测框的位置。

设 feature maps 的尺寸为 W x H , 那么总共有 个锚框 W x H x 9 个锚框。



## RPN 中的卷积

当我们已经得到锚框(anchor boxes)后，需要完成以下两项任务：

- (1) anchor box 中是否包含识别物体，foreground/background，二分类问题。
- (2) 如果 anchor box 包含物体，怎么调整，才能使得 anchor box 与 ground truth 更接近。

### 总结：

设 backbone 输出的 feature maps 的尺寸为  $W \times H \times C$ ，设置了  $W \times H \times k$  个 anchor boxes，则上面的卷积神经网络的输出为：

- 大小为  $W \times H \times k \times 2$  的 positive/negative Softmax 分类矩阵，可记为 `rpn_cls_score`；
- 大小为  $W \times H \times k \times 4$  的 Bounding Box Regression 坐标偏移矩阵，也就是在原始锚框的基础上做的修改和缩放  $[d_x(A), d_y(A), d_w(A), d_h(A)]$ ，可记为 `rpn_bbox_pred`；

## Bounding Box Regression 的原理

给定 anchor  $A = (A_x, A_y, A_w, A_h)$  和  $GT = (G_x, G_y, G_w, G_h)$ ，找到一组变换：

$$[d_x(A), d_y(A), d_w(A), d_h(A)] \quad s.t. \quad (G'_x, G'_y, G'_w, G'_h) \approx (G_x, G_y, G_w, G_h)$$

比较简单的思路：

- 先做平移

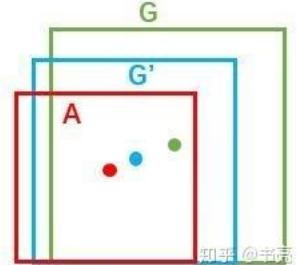
$$G'_x = A_w \cdot d_x(A) + A_x$$

$$G'_y = A_h \cdot d_y(A) + A_y$$

- 再做缩放

$$G'_w = A_w \cdot \exp(d_w(A))$$

$$G'_h = A_h \cdot \exp(d_h(A))$$



## RPN 中的 Proposal

Proposal 的输入有三个，分别是：

- softmax 分类矩阵 `rpn_cls_score`。
- Bounding Box Regression 坐标矩阵 `rpn_bbox_pred`。
- `img_info`：设输入图片尺寸为  $P \times Q$ ，在 Faster-RCNN 的预处理中，会 reshape 为  $M \times N$ ，则 `img_info` =  $[M, N, scale\_factor]$  保存了缩放的信息。

从而 RPN 的输出为：

- `rpn_rois`: RPN 产生的 ROIs (Region of Interests)。
- `rpn_roi_probs`: 表示 ROI 包含物体的概率。

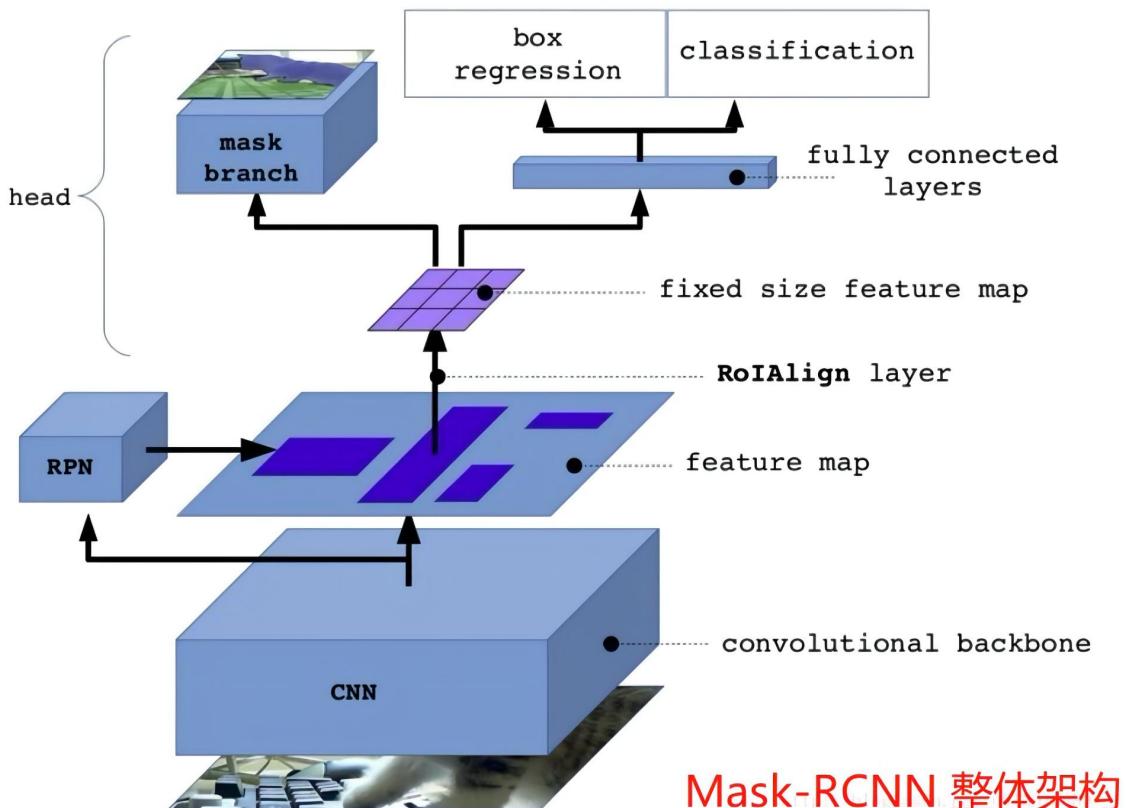
简而言之，RPN 只挑选出了可能包含物体的区域(`rpn_rois`)以及其包含物体的概率(`rpn_roi_probs`)。在后续处理中，设定一个阈值 `threshold`，如果某个 ROI 的 `rpn_roi_probs` > `threshold`，则再判定其类别。如果某个 ROI 的 `rpn_roi_probs` < `threshold`，则忽略。

### 总结：

# Mask RCNN 模型图

## 参考文献:

1. <https://arxiv.org/abs/1409.1556>
2. matterport/Mask\_RCNN: [https://github.com/matterport/Mask\\_RCNN](https://github.com/matterport/Mask_RCNN)
3. Mask-RCNN 算法及其实现详解: <https://blog.csdn.net/remanented/article/details/79564045>



class MaskRCNN() 成员函数 `def build(self, mode, config)` 的功能是 # Build Mask R-CNN architecture, 其构成为:

- # Inputs
- # Build the shared convolutional layers.
  - # Bottom-up Layers ==>  $\_$ , C2, C3, C4, C5
  - # Top-down Layers ==> P2, P3, P4, P5, P6
  - # Anchors
  - # RPN Model
  - # Generate proposals
- if mode == "training":
  - # Generate detection targets. Subsamples proposals and generates target outputs for training.
  - # Network Heads
  - # Losses
  - # Model
- else:
  - # Network Heads. Proposal classifier and BBox regressor heads.
  - # Detections
  - # Create masks for detections

# Training

流程图

class MaskRCNN() 里的 def build(self, mode, config) # Build Mask R-CNN architecture.

class MaskRCNN() 里的  
def get\_anchors(self, image\_shape)

## Anchor Generator

核心函数:  
utils.generate\_pyramid\_anchors()  
调用了 utils.generate\_anchors()

anchors

anchors: normalized coordinates.

## backbone

(ResNet101)

特征融合

rpn\_feature\_maps  
= [P2, P3, P4, P5, P6]

mrcnn\_feature\_maps  
= [P2, P3, P4, P5]

## RPN

核心函数:  
build\_rpn\_model() 里调用了  
rpn\_graph()

rpn\_class\_logits,  
rpn\_class,  
rpn\_bbox

rpn\_bbox: Deltas to be applied to anchors.

## ProposalLayer(KE.Layer)

成员函数 call() 里调用了:

- clip\_boxes\_graph(boxes, window)
- nms(boxes, scores) # Non-max suppression
- config.RPN\_NMS\_THRESHOLD = 0.7

rpn\_rois

## Losses

- rpn\_class\_loss\_graph()
- rpn\_bbox\_loss\_graph()
- mrcnn\_class\_loss\_graph()
- mrcnn\_bbox\_loss\_graph()
- mrcnn\_mask\_loss\_graph()

## DetectionTargetLayer(KE.Layer)

成员函数 call() 里调用了:  
def detection\_targets\_graph( proposals, # 即 rpn\_rois  
gt\_class\_ids,  
gt\_boxes,  
gt\_masks,  
config)

rois,  
target\_class\_ids,  
target\_bbox,  
target\_mask

output\_rois

= KL.Lambda(lambda x: x \* 1, name="output\_rois")(rois)

mrcnn\_feature\_maps  
= [P2, P3, P4, P5]

rois

## fpn\_classifier\_graph()

使用 class PyramidROIAlign(KE.Layer)

## build\_fpn\_mask\_graph()

使用 class PyramidROIAlign(KE.Layer)

rpn\_class\_loss,  
rpn\_bbox\_loss,  
class\_loss,  
bbox\_loss,  
mask\_loss

mrcnn\_class\_logits,  
mrcnn\_class,  
mrcnn\_bbox

mrcnn\_mask

注:

- 红色字体 inputs 为 模型的输入。
- 其它红色字体 (粉红色除外) 为模型的输出。

# Predict

流程图

class MaskRCNN() 里的 def build(self, mode, config) # Build Mask R-CNN architecture.

class MaskRCNN() 里的  
def get\_anchors(self, image\_shape)

## Anchor Generator

核心函数:  
utils.generate\_pyramid\_anchors()  
调用了 utils.generate\_anchors()

**anchors**

**anchors**: normalized coordinates.

## backbone

(ResNet101)

特征融合

rpn\_feature\_maps  
= [P2, P3, P4, P5, P6]

mrcnn\_feature\_maps  
= [P2, P3, P4, P5]

## RPN

核心函数:  
build\_rpn\_model() 里调用了  
rpn\_graph()

rpn\_class\_logits,  
rpn\_class,  
rpn\_bbox

rpn\_bbox: Deltas to be applied to anchors.

## ProposalLayer(KE.Layer)

成员函数 call() 里调用了:

- clip\_boxes\_graph(boxes, window)
- nms(boxes, scores) # Non-max suppression
- config.RPN\_NMS\_THRESHOLD = 0.7

rpn\_rois

## fpn\_classifier\_graph()

使用了 class PyramidROIAlign(KE.Layer)

mrcnn\_class\_logits,  
mrcnn\_class,  
mrcnn\_bbox

## DetectionLayer(KE.Layer)

成员函数 call() 里调用了:

```
def refine_detections_graph(  
    rpn_rois,  
    mrcnn_class,  
    mrcnn_bbox, # deltas  
    window,  
    config)
```

且内嵌调用 def nms\_keep\_map() # Non-Maximum Suppression

detections

[ batch, num\_detections, (y1, x1, y2, x2, class\_id, score) ]  
in normalized coordinates.

detection\_boxes  
= KL.Lambda(lambda x: x[...,:4])(detections)

mrcnn\_feature\_maps  
= [P2, P3, P4, P5]

## build\_fpn\_mask\_graph()

使用了 class PyramidROIAlign(KE.Layer)

mrcnn\_mask

注:

- 红色字体 inputs 为 模型的输入。
- 其它红色字体 (粉红色除外) 为模型的输出。

# 其它

## 参考文献：

- 完整代码+实操！手把手教你操作Faster R-CNN和Mask R-CNN：  
<https://blog.csdn.net/dQCFKyQDXYm3F8rB0/article/details/89007726>
- RCNN、FastRCNN、FasterRCNN、MaskRCNN目标检测：  
[https://blog.csdn.net/weixin\\_43877335/article/details/124678384](https://blog.csdn.net/weixin_43877335/article/details/124678384)

机器视觉领域的核心问题之一就是**目标检测（Object Detection）**，它的任务是找出图像当中所有感兴趣的目标（物体），确定其位置和大小。当下非常火热的无人驾驶汽车，就非常依赖目标检测和识别，这需要非常高的检测精度和定位精度。

作为经典的目标检测框架 **Faster R-CNN**，虽然是2015年的论文，但是它至今仍然是许多目标检测算法的基础，这在飞速发展的深度学习领域十分难得。而在 Faster R-CNN 的基础上改进的 **Mask R-CNN** 在2018年被提出，并斩获了ICCV2017年的最佳论文。Mask R-CNN可以应用到**人体姿势识别**，并且在**实例分割**、**目标检测**、**人体关键点检测**三个任务都取得了很好的效果。

因此，一些深度学习框架如百度PaddlePaddle 开源了用于目标检测的 RCNN 模型，从而可以快速构建满足各种场景的应用，包括但不仅限于**安防监控**、**医学图像识别**、**交通车辆检测**、**信号灯识别**、**食品检测**等等。

目前，用于目标检测的方法通常属于**基于机器学习的方法**或**基于深度学习的方法**。

- 对于机器学习方法，首先使用 SIFT、HOG 等方法定义特征，然后使用支持向量机（SVM）、Adaboost 等技术进行分类。
- 对于深度学习方法，深度学习技术能够在没有专门定义特征的情况下进行端到端目标检测，并且通常基于卷积神经网络（CNN）。

但是传统的目标检测方法有如下几个问题：光线变化较快时，算法效果不好；缓慢运动和背景颜色一致时不能提取出特征像素点；时间复杂度高；抗噪性能差。

因此，**基于深度学习的目标检测方法**得到了广泛应用，该框架包含有 **Faster R-CNN**, **Yolo**, **Mask R-CNN**等。

## 从 R-CNN 到 Mask R-CNN

Mask R-CNN 是承继于 Faster R-CNN，Mask R-CNN 只是在 Faster R-CNN 上面增加了一个 Mask Prediction Branch (Mask预测分支)，并且在 ROI Pooling 的基础之上提出了 ROI Align。所以要想理解 Mask R-CNN，就要先熟悉 Faster R-CNN。同样的，Faster R-CNN 是承继于 Fast R-CNN，而 Fast R-CNN 又承继于 R-CNN，因此，为了能让大家更好的理解**基于 CNN 的目标检测方法**，我们从 R-CNN 开始切入，一直介绍到 Mask R-CNN。

总结：

# R-CNN

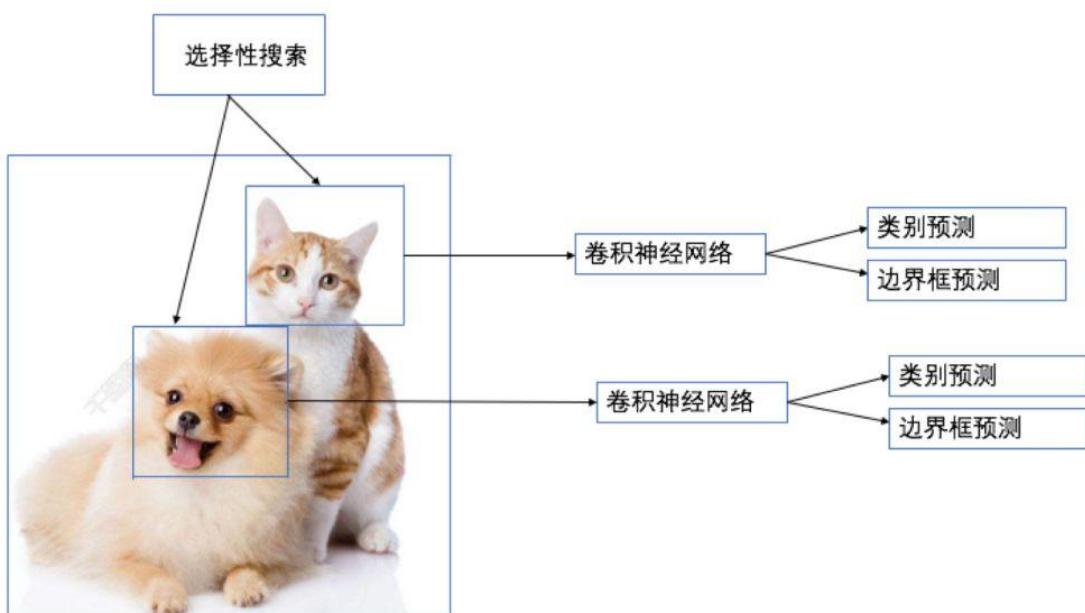
区域卷积神经网络 (Regions with CNN features) 使用深度模型来解决目标检测。

## R-CNN的操作步骤

- **Selective search (选择性搜索)**：首先对每一张输入图像使用选择性搜索来选取多个高质量的提议区域 (region proposal)，大约提取2000个左右的提议区域。
- **Resize (图像尺寸调整)**：接着对每一个提议区域，将其缩放 (warp) 成卷积神经网络需要的输入尺寸 (277\*277)。
- **特征抽取**：选取一个预先训练好的卷积神经网络，去掉最后的输出层来作为特征抽取模块。
- **SVM (类别预测)**：将每一个提议区域提出的 CNN 特征输入到支持向量机 (SVM) 来进行物体类别分类。注：这里第  $i$  个 SVM 用来预测样本是否属于第  $i$  类。
- **Bounding Box Regression (边框预测)**：对于支持向量机分好类的提议区域做边框回归，训练一个线性回归模型来预测真实边界框，校正原来的建议窗口，生成预测窗口坐标。

## R-CNN优缺点分析

- **优点**：R-CNN 对之前物体识别算法的主要改进是使用了预先训练好的卷积神经网络来抽取特征，有效的提升了识别精度。
- **缺点**：速度慢。对一张图像我们可能选出上千个兴趣区域，这样导致每张图像需要对卷积网络做上千次的前向计算。



## R-CNN 训练阶段：

CNN 训练是在 imagenet 上预训练好的，迁移进行 fine-tuning 训练，将最后一层替换为 N类+1背景类 输出。

- fine-tuning 时正负样本选择：将与 ground-truth 的 IoU $\geq 0.5$  的 proposal 作为正样本，不分类别，剩下作为负样本，每个 batch 中正负样本比例是1:3。
- N 个 SVM 训练时正负样本选择：正样本是 ground-truth，负样本是 IoU < 0.3 的 proposal，忽略 > 0.3 的 proposal，采用 hard negative mining method 加快训练。
- Bounding box regression 训练，输入是 IoU > 0.6 的 proposal，label 是ground-truth，预测的是 dx(P), dy(P), dw(P) 和 dh(P) 这四个缩放因子，损失函数如式 (2) 所示。P 对应的是候选框的中心坐标以及框的高宽，G 对应的是最终预测的边界框的中心坐标以及宽高。

$$\begin{aligned}\hat{G}_x &= P_w d_x(P) + P_x \\ \hat{G}_y &= P_h d_y(P) + P_y \\ \hat{G}_w &= P_w \exp(d_w(P)) \\ \hat{G}_h &= P_h \exp(d_h(P)). \\ t_x &= \frac{(G_x - P_x)}{P_w} \\ t_y &= \frac{(G_y - P_y)}{P_h} \\ t_w &= \log\left(\frac{G_w}{P_w}\right) \\ t_h &= \log\left(\frac{G_h}{P_h}\right)\end{aligned}\tag{1}$$

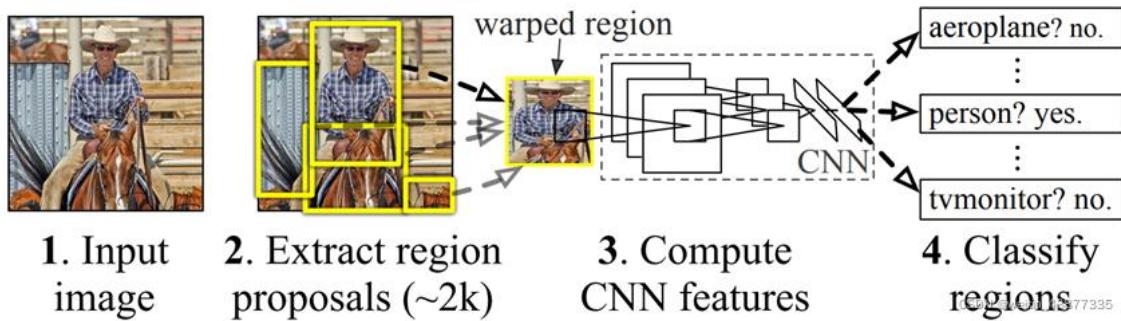
$$w_* = \underset{\hat{w}_*}{\operatorname{argmin}} \sum_i^N \left( t_x^i - \hat{w}_*^T \phi_5(P^i) \right)^2 + \lambda \|\hat{w}_*\|^2 \tag{2}$$

CSDN @weixin\_43877335

## R-CNN 测试阶段

输入一张图像，采用 Selective Search 方法生成约 2k 个候选框 region proposal，将 proposal 缩放到统一大小，输入到 CNN 中提取 proposal 的 feature，再采用 SVM 对 feature 进行分类，判别 feature 所属类别的置信度，采用非极大值抑制 (NMS,non-maximum suppression) 方法剔除重复预测的 proposal，然后将剩下的 proposal 的 feature 输入至 linear regression 来修正候选框的位置，得到预测框 bouding box。

### R-CNN: Regions with CNN features



## Fast R-CNN

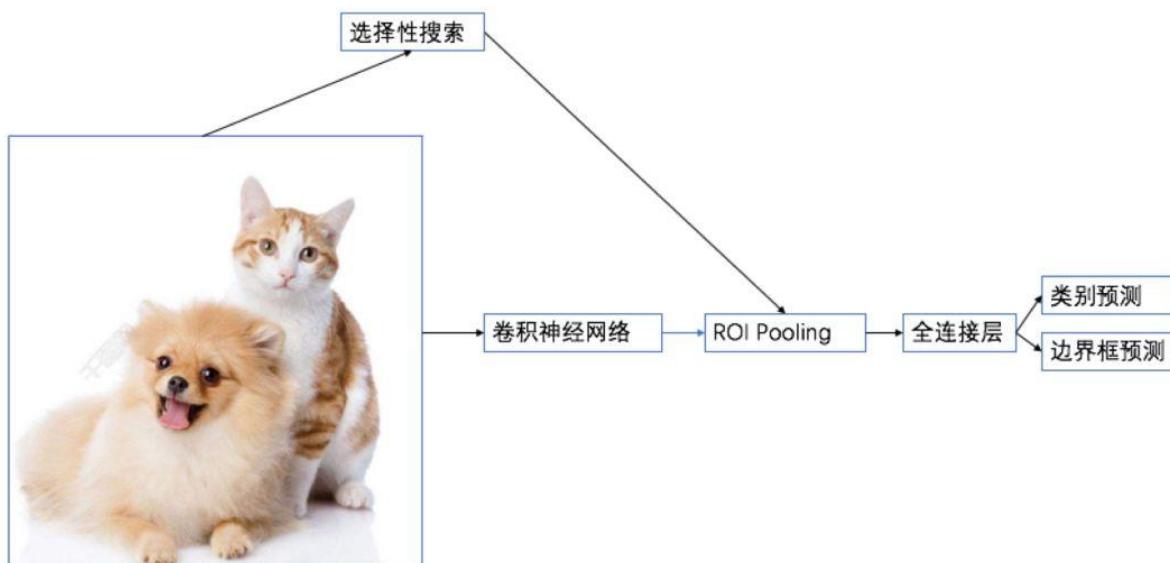
R-CNN 的主要性能瓶颈在于需要对每个提议区域（region proposal）独立的抽取特征，这会造成区域会有大量重叠，独立的特征抽取导致了大量的重复计算。因此，Fast R-CNN 对 R-CNN 的一个主要改进在于首先对整个图像进行特征抽取，然后再选取提议区域，从而减少重复计算。

### Fast R-CNN的操作步骤

- Selective Search（选择性搜索）：首先对每一张输入图像使用选择性搜索（selective search）算法来选取多个高质量的提议区域（region proposal），大约提取2000个左右的提议区域。
- 将整张图片输入卷积神经网络，**对全图进行特征提取**。
- 把提议区域**映射**到卷积神经网络的最后一层卷积（feature map）上。
- ROI Pooling：引入了兴趣区域池化层（Region of Interest Pooling）来对每个提议区域提取同样大小的输出。
- **Softmax**：在物体分类时，Fast R-CNN 不再使用多个 SVM，而是像之前图像分类那样使用 Softmax 回归来进行多类预测。

### Fast R-CNN优缺点分析

- **优点**：对整个图像进行特征抽取，然后再选取提议区域，从而减少重复计算。
- **缺点**：选择性搜索费时；**不用Resize,不适合求导**。



## Fast R-CNN 训练阶段：

Fast-RCNN 实现了端到端训练。采用 Selective Search 算法在一张图像上生成约 2k 个 proposal，然后经过正负样本采样获得合适的 proposal，将图像输入网络得到 feature map，将图像上的 proposal 投影到 feature map 获得对应的特征矩阵，特征矩阵通过 ROI pooling 层，ROI pooling 层是将输入的 proposal 划分为  $7 \times 7$  大小的块，在每个块中进行 maxpooling，这样可以不限制输入图像的尺寸。然后经过一系列的 FC 层，通过 softmax 进行分类，类别为 N+1，bbox regression 修正候选框的位置，输出为 N 个类别分别对应的回归参数，共  $4N$  个。

正负样本采样：在每个 batch 中包含 2 张图像和 128 个 proposal，每张图像有 64 个 proposal，挑选 batch 中 25% 的 proposal 作为正样本，这些 proposal 的 IoU > 0.5。

式(3)是模型的整体损失。式(4)是分类损失，也是交叉熵损失。式(5)是 bbox regression 损失， $[u \geq 1]$  代表如果是正样本， $u=1$ ，如果是负样本， $u=0$

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v) \quad (3)$$

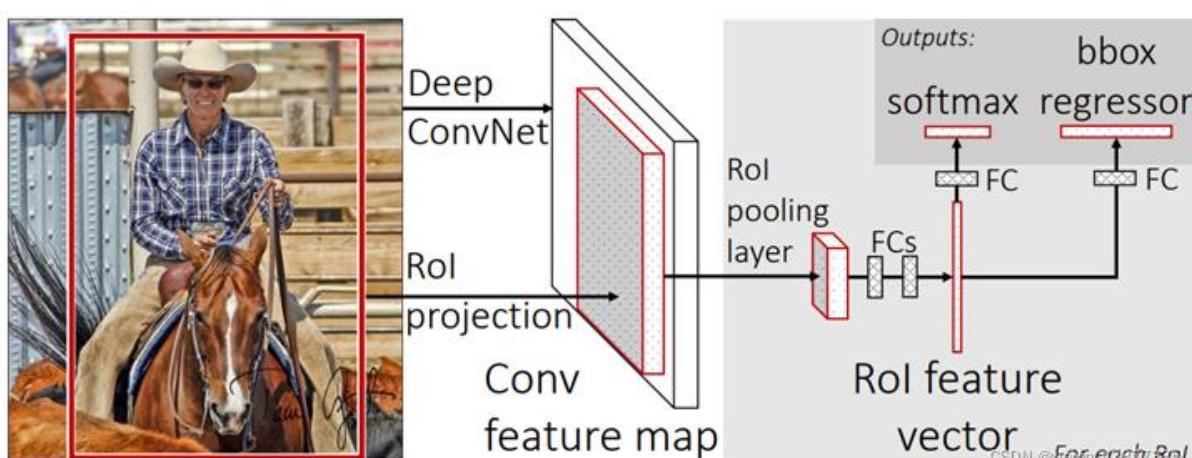
$$L_{cls}(p, u) = -\log P_u \quad (4)$$

$$L_{loc}(t^u, v) = \sum_{i \in \{x, y, w, h\}} \text{smooth}_{L_1}(t_i^u - v_i) \quad (5)$$
$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

CSDN @weixin\_43877335

## Fast R-CNN 测试阶段

通过 SS 在图像上生成 2K 个 proposal，图像输入网络得到 feature map，proposal 映射到 feature map 得到对应的特征矩阵，再通过 ROI pooling 和一系列 FC 得到预测的类别和回归修正值，然后通过 NMS 得到对 proposal 进行筛选，得到 bounding box



## Faster R-CNN

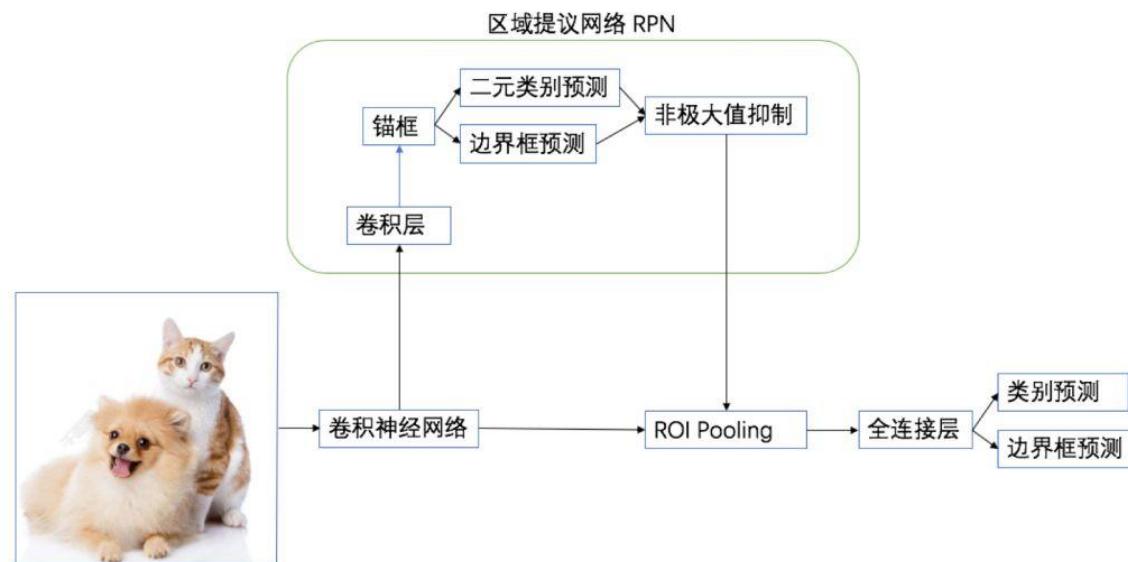
Faster R-CNN 对 Fast R-CNN 做了进一步改进，它将 Fast R-CNN 中的选择性搜索替换成了区域提议网络（region proposal network, RPN）。RPN 以锚框（anchors）为起始点，通过一个神经网络来选择区域提议。

Faster R-CNN 整体网络可以分为4个主要内容：

- **基础卷积层 (CNN)**：作为一种卷积神经网络目标检测方法，Faster R-CNN首先使用一组基础的卷积网络提取图像的特征图。特征图被后续RPN层和全连接层共享。本示例采用ResNet-50作为基础卷积层。
- **区域生成网络(RPN)**：RPN 网络用于生成候选区域 (proposals)。该层通过一组固定的尺寸和比例得到一组锚点 (anchors)，通过 softmax 判断锚点属于前景或者背景，再利用区域回归修正锚点从而获得精确的候选区域。
- **RoI Pooling**：该层收集输入的特征图和候选区域，将候选区域映射到特征图中并池化为统一大小的区域特征图，送入全连接层判定目标类别，该层可选用 RoIPool 和 RoIAvg 两种方式，在config.py中设置roi\_func。
- **检测层**：利用区域特征图计算候选区域的类别，同时再次通过区域回归获得检测框最终的精确位置。

Faster R-CNN优缺点分析

- **优点**：RPN 通过标注来学习预测跟真实边界框更相近的提议区域，从而减小提议区域的数量同时保证最终模型的预测精度。
- **缺点**：无法达到实时目标检测。



## Faster R-CNN

Faster-RCNN 和 Fast-RCNN 的区别就在于 proposal 的选取，Faster-RCNN 用 RPN 来代替 SS 算法选取 proposal。Faster-RCNN 的训练过程是先将图像输入网络得到 feature map，使用 RPN 生成 proposal，将 proposal 投影到 feature map 获得对应的特征矩阵，特征矩阵通过 ROI pooling 层，ROI pooling 层是将输入的 proposal 划分为 $7 \times 7$ 大小的块，在每个块中进行 maxpooling，这样可以不限制输入图像的尺寸。然后经过一系列的 FC 层，通过 softmax 进行分类，类别为  $N+1$ ，bbox regression 修正候选框的位置，输出为  $N$  个类别分别对应的回归参数，共  $4N$  个。

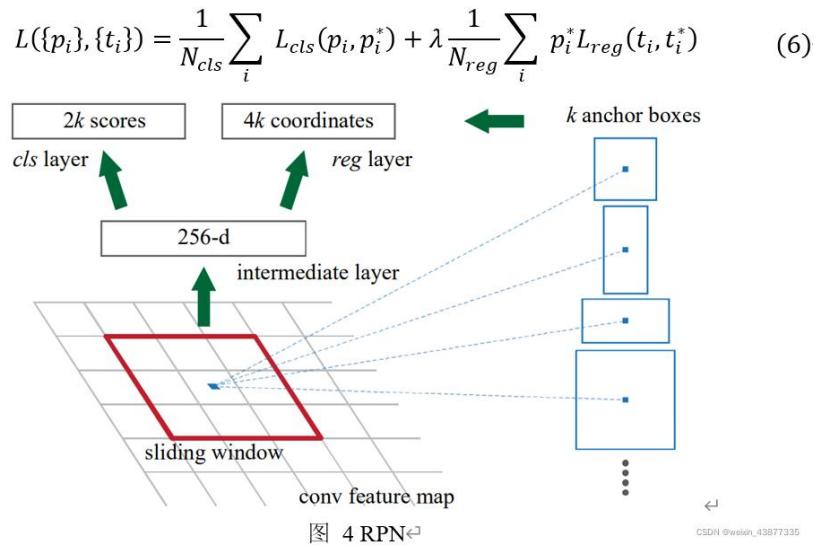
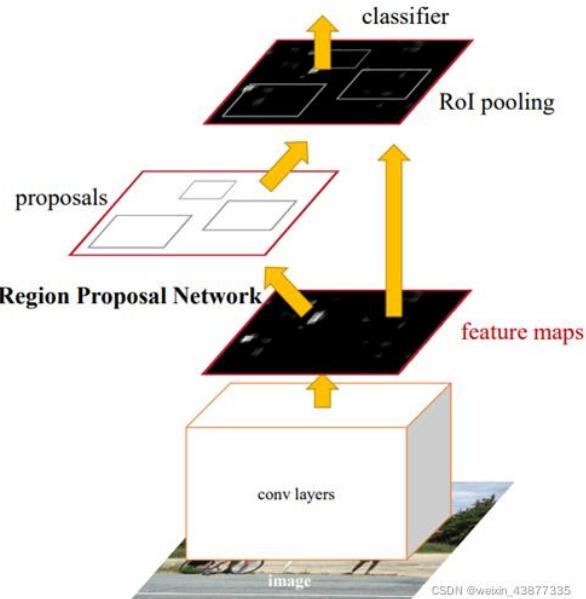


图 4 RPN

RPN 是先以 feature map 每个像素对应的原图上的像素为中心得到  $3 \times 3$  个不同的 anchor，通过正负样本采样，一个图像得到  $k$  个 anchor，通过 CNN 与 feature map 卷积得到大小不变通道数减少的特征向量，每个向量长度是通道个数，将所有特征向量经过 FC 层，再通过 cls layer ( $1 \times 1$  的  $2k$  个卷积) 得到  $k$  个 anchor 对应的前景/背景的置信率，一共是  $2k$  个，再通过 reg layer ( $1 \times 1$  的  $4k$  个卷积) 得到每个 anchor 的 4 个回归偏置，损失函数如式(6)，根据偏置调整后的 anchor 是得到的 proposal。在测试部分，通过 NMS 再进一步筛选 proposal。

正负样本采样：

每张图片采样 256 个 anchor，正负样本比例是 1:1，与 ground-truth 的 IoU 最大的 anchor 和  $\text{IoU} > 0.7$  的 anchor 是正样本， $\text{IoU} < 0.3$  的 anchor 是负样本，剩下的忽略。

# Mask R-CNN

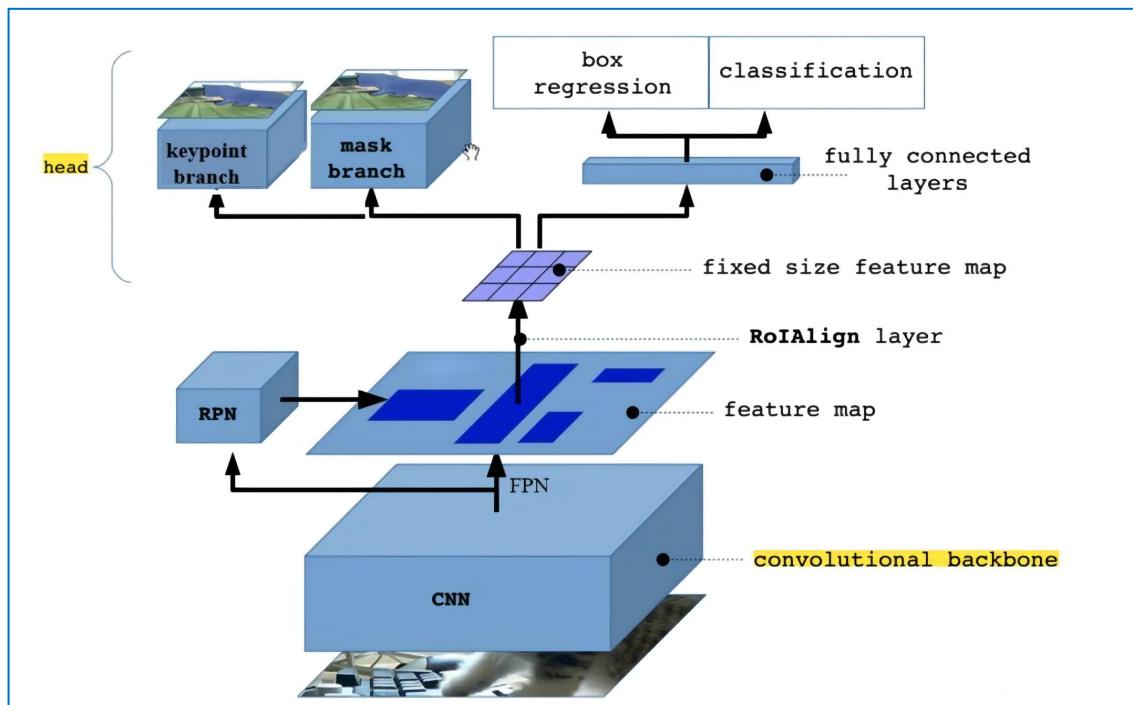
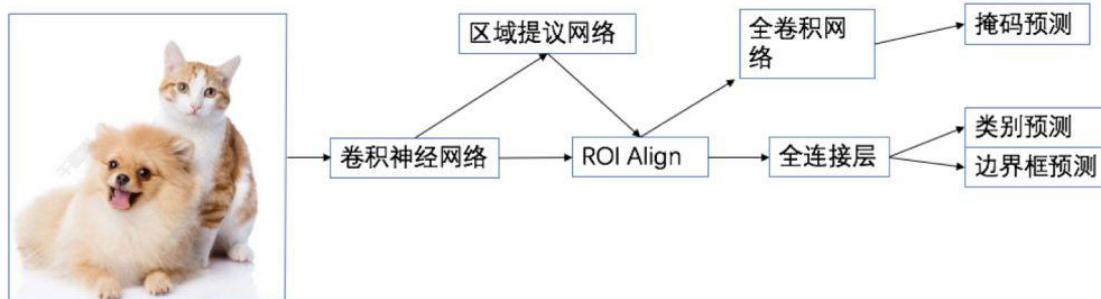
Faster R-CNN 在物体检测中已达到非常好的性能， Mask R-CNN在此基础上更进一步：得到像素级别的检测结果。对每一个目标物体，不仅给出其边界框，并且对边界框内的各个像素是否属于该物体进行标记。

Mask R-CNN同样为**两阶段框架**：

- 第一阶段扫描图像生成候选框；
- 第二阶段根据候选框得到分类结果，边界框，同时在原有Faster R-CNN模型基础上添加分割分支，得到掩码结果，实现了掩码和类别预测关系的解耦。

## Mask R-CNN 的创新点

- **解决特征图与原始图像上的 RoI 不对准问题**：在Faster R-CNN中，没有设计网络的输入和输出的**像素级别的对齐机制 (pixel to pixel)**。为了解决特征不对准的问题，文章作者提出了 RoIAxis 层来解决这个问题，它能准确的保存空间位置，进而提高 mask 的准确率。
- **将掩模预测 (mask prediction) 和分类预测 (class prediction) 拆解**：该框架结构对每个类别独立的预测一个二值 mask，不依赖分类 (classification) 分支的预测结果。
- **掩模表示 (mask representation)**：有别于类别，框回归，这几个的输出都可以是一个向量，但是 mask 必须要保持一定的空间结构信息，因此作者采用全连接卷积层 (FCN) 对每一个RoI中预测一个  $m^*m$  的掩模。



## Mask-RCNN:

Mask-RCNN 是在 Faster-RCNN 上的改进，增加了一个 mask 分支，mask 部分采用的是 FCN 网络做分割。但是 mask 部分的输出是 K 个 binary mask，二分类采用的是 sigmoid 函数，并不预测所属类别，所属类别的预测是依赖于 Faster RCNN 的分类部分。

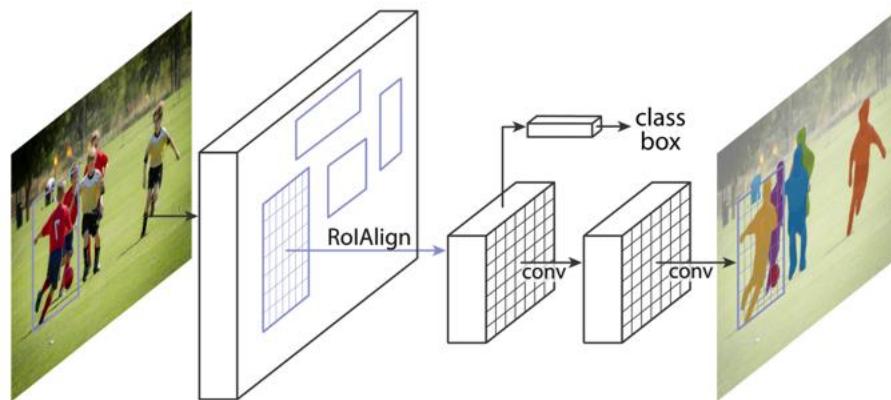


图 Mask RCNN

CSDN @weixin\_43877335

损失函数如下式。

$$L = L_{cls} + L_{box} + L_{mask_{\leftarrow}}$$

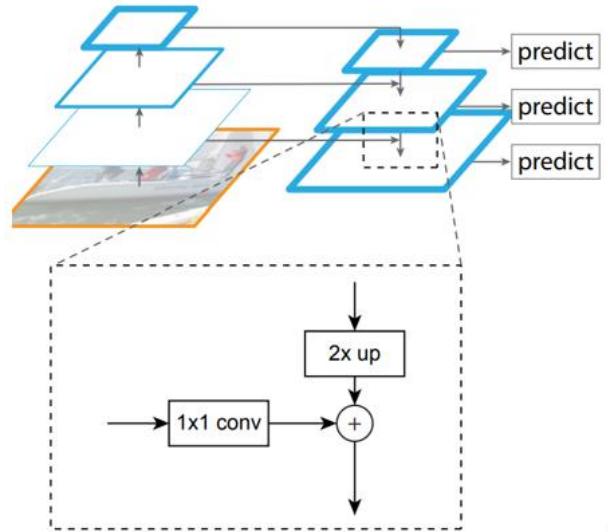
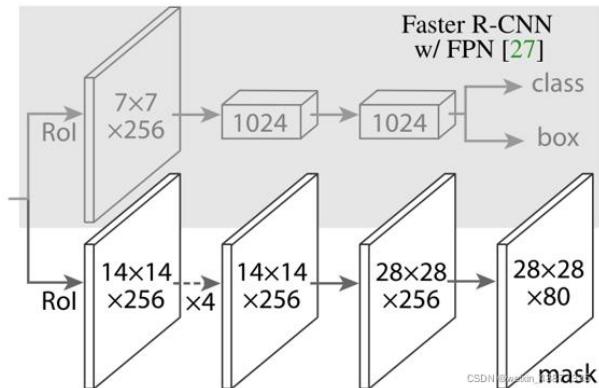


图 FPN

CSDN @weixin\_43877335

模型采用的是 FPN 架构，可以识别多尺度的物体。论文采用了 RoIAgnostic 是因为 RoIPooling 会出现 misalignment, RoIAgnostic 通过双线性插值可以得到准确的空间定位。