Mask RCNN

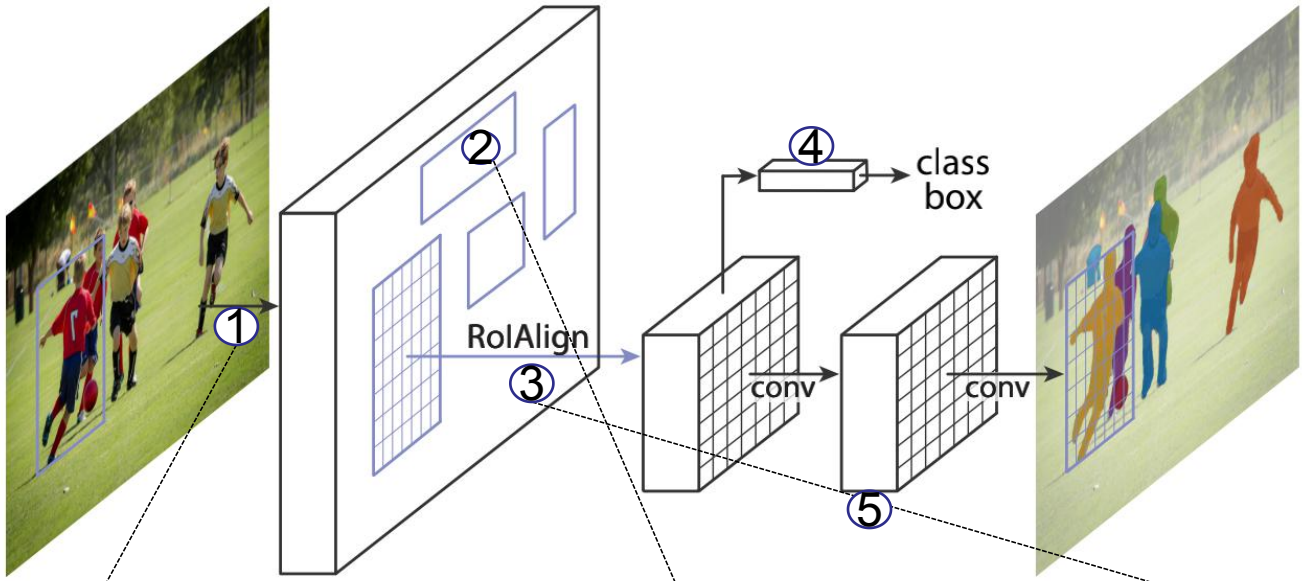
一、论文解读
- 论文地址: https://arxiv.org/pdf/1703.06870.pdf
  （一）、网络整体结构
  （二）、rpn网络
  （三）、RoIAlign层
  （四）、bbox, class, mask预测

二、开源代码解读
- 代码地址: https://github.com/matterport/Mask_RCNN
  （一）、代码整体概况
  （二）、模型输入
  （三）、resnet,top_down layer
  （四）、rpn及其target
  （五）、mask-rcnn及其target
  （六）、loss

三、复现
- 代码地址:
  （一）、backbone骨架
  （二）、rpn及其target
  （三）、mask-rcnn及其target
  （四）、loss
  （五）、pascal voc数据集训练
  （六）、coco数据集训练
  （七）、自定义数据集训练

一、论文解读

（一）、网络整体结构



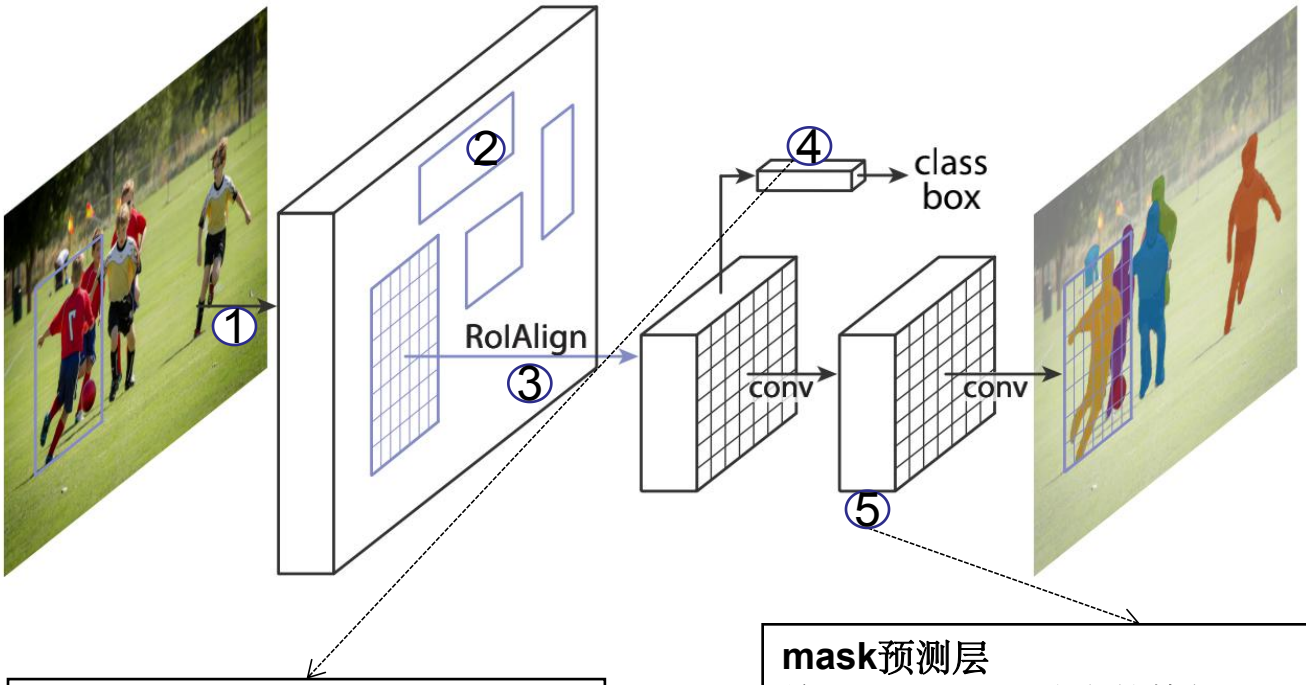
卷积层vgg/resnet
输入：任意尺度图片
输出：卷积层5个中间层输出，分别为原图片尺度的（1/4, 1/4, 1/8, 1/16, 1/32），再对这些特征输出经过进一步上采样，卷积合并，最终输出6个尺度特征层，分别为原图大小的（1/4, 1/8, 1/16, 1/32, 1/64）

rpn层
输入：卷积层输出的5个尺度特征
输出：在每个特征层生成一系列的边框偏移缩放量和前景背景类别预测，大小分别为：
bbox=(batch, h, w, 3*4)
class=(batch, h, w,3*2)
其中3表示每个像素点生成3种不同旋转比例的边框，4表示边框4个平移缩放量，2表示前景背景类别

RoIAlign层
输入：rpn网络输出的proposal，以及卷积层输出的5个尺度特征
输出：计算不同proposal对应到不同尺度下的特征，利用proposal对该特征进行裁剪，resize，池化提取特征，最后大小为：[batch, num_rois, pool_size, pool_size, channels]

一、论文解读

（一）、网络整体结构



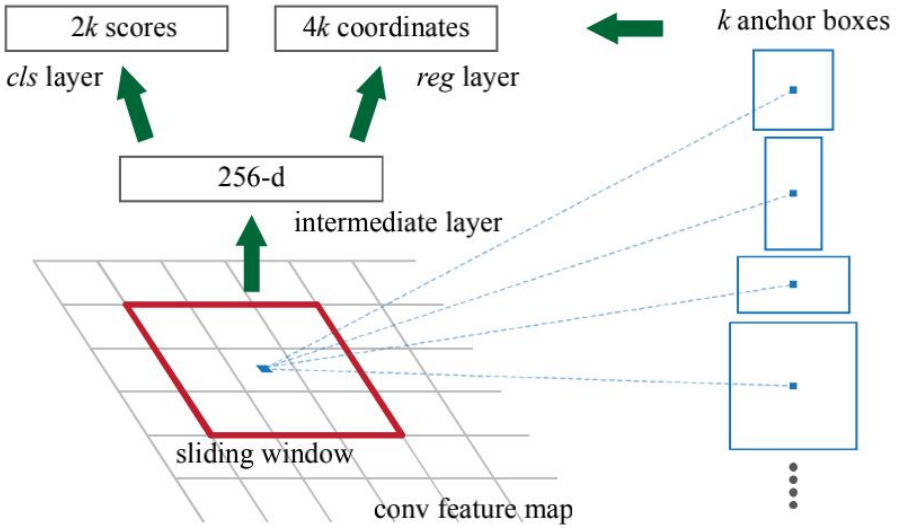
mask-rcnn类别边框预测层
输入：RoIAlign输出的特征
输出：对RoIAlign输出的特征进行一些列的卷积，最后对边框预测输出为：
[batch, num_rois, num_classes * (dy, dx, log(dh), log(dw))]
对类别预测输出为：
[batch, num_rois,num_classes]

mask预测层
输入：RoIAlign输出的特征
输出：对RoIAlign输出的特征进行一些列的卷积，转置卷积，最后输出mask的预测为：
[batch, num_rois, pool_size*2, pool_size*2, num_classes]

注意一点：
mask类别边框预测层和mask预测层中的RoIAlign是互相独立计算的，因此最终的RoIAlign输出pool shape是不同的，分别为7和14

一、论文解读

（二）、rpn网络



RPN网络主要输出两个东西：

1. roi, 对应在特征层每个特征点产生4*k个变量, 其中4表示[dy, dx, dh, dw]四个边框平移缩放量, k表示个边框

2. scores, 对应在特征层每个特征点产生2*k个变量, 其中2表示前景和背景概率， k表示3个边框

一、论文解读

（三）、RoIAlign (roi level)

FPN论文中的RoI Level校准：

$$k = \lfloor k_0 + \log_2(\sqrt{wh}/224) \rfloor .$$

出处：《Feature Pyramid Networks for Object Detection》

Mask-RCNN论文中采用的是下面的公式：

$$k = \left\lfloor k_0 + log_2\left(\sqrt{wh}/(224/\sqrt{image\ area})\right) \right\rfloor$$

关于此公式的两个重要点：

1.由于mask-rcnn论文训练数据的box,anchor都做了规整,所以roi level的计算部分也需要224/sqrt(image area)

2.计算得到的k即为roi对应的level，level一共有4个，
level=2表示映射回特征P2，大小为原输入图片的1/4
level=3表示映射回特征P3，大小为原输入图片的1/8
level=4表示映射回特征P4，大小为原输入图片的1/16
level=5表示映射回特征P5，大小为原输入图片的1/32

一、论文解读

（三）、RoIAlign (roi level)

$$k = \left\lfloor k_0 + log_2\left(\sqrt{wh}/(224/\sqrt{image\ area})\right)\right\rfloor$$

计算例子：

k_0=4(论文里的初始值)

假设:
w=0.25 (表示原图大小1/4的宽)
h=0.25 (表示原图大小1/4的高)
image area = 448^2 (表示448*448图片)

那么:
t = log_2(0.25 / (224/ 448)) = -1
k = k_0 + t = 3(即对应回特征层P3)

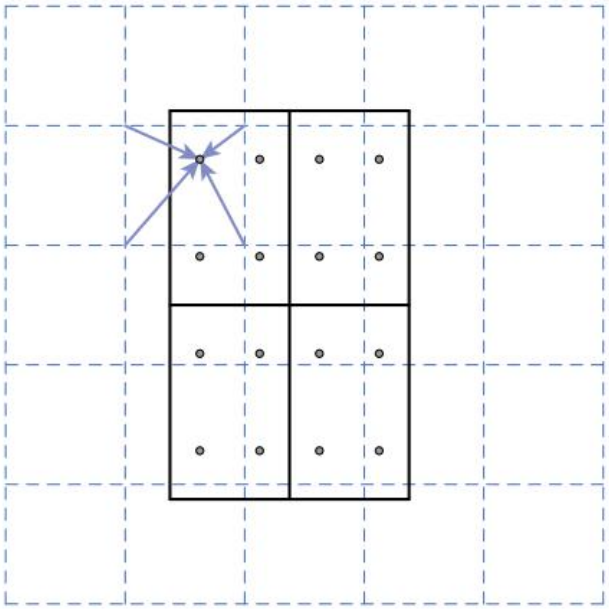
对于非整数的计算：
k = minimum(5, maximum(2, k_0 + t)))

需要注意的是:
224这个值实际上可以根据自己的数据情况进行调整，比如笔者的输入数据是640^2，为了让roi=0.5^2的数据能去P4层面采样裁剪，可以把224替换成320.

一、论文解读

（三）、RoIAlign (crop resize)

RoI crop resize(双向线性插值):



crop resize操作： 主要是解决了RoI小数问题

假设:
RoI = [1.4, 0.95, 3.25, 4.15]
对应图中黑色边框[xmin, ymin, xmax, ymax]

那么小数是无法裁剪出特征层上对应特征区域的,
在Rcnn中使用RoIPooling是直接将小数规整。

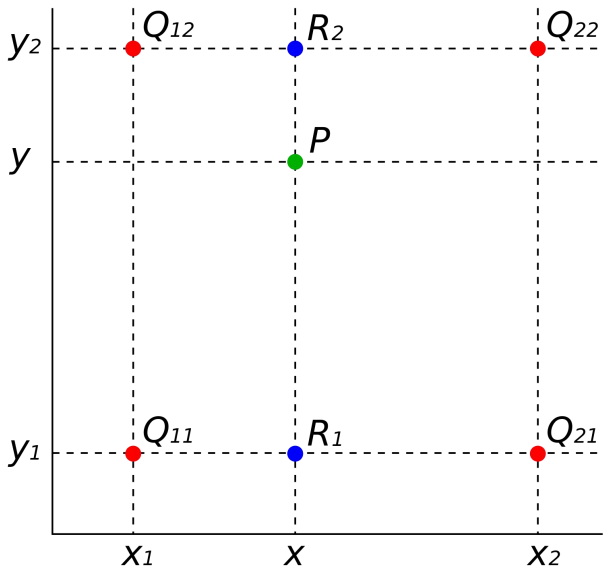
在RoIAlign中为了将上述RoI resize成2x2尺寸, 具体为:

1. 将RoI分成2x2 4个bin(论文词汇)
2. 在每个bin中去4个点(作者兼顾性能和效果的选择),
如图黑点
3. 每个bin中4个点, 每个点对应到特征层上最近的4个,
采用双向线性插值得到该点的具体值
4. bin中4个点取最大值,作为bin本身的值, 如此得到4
个bin的值， 即为resize后的值

一、论文解读

（三）、RoIAlign (crop resize)

RoI crop resize(双向线性插值):



双向线性插值:

假设各点坐标为:
$x_1 = [1,0]$, $x_2 = [2,0]$, $y_1 = [0, 1]$, $y_2 = [0,2]$
$x = [1.4,0]$, $y = [0,1.7]$
$P = [1.4, 1.7]$

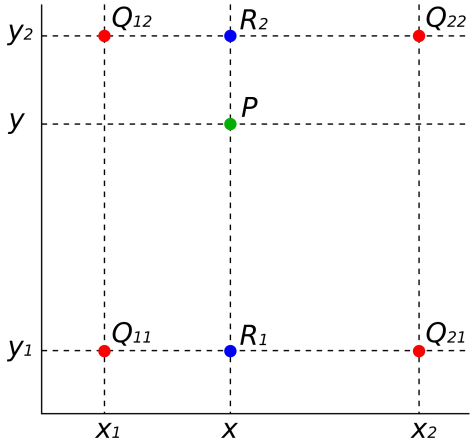
4个像素点Q_{11},Q_{21},Q_{12},Q_{22}的值分别为:
100,105,110,112

插值过程可以先在X方向上插值两次得到R1,R2
再在Y方向上插值一次得到P

一、论文解读

（三）、RoIAlign (crop resize)

双向线性插值例子：



$$f(R_1) = \frac{x_2 - x}{x_2 - x1} Q_{11} + \frac{x - x_1}{x_2 - x_1} Q_{21} = \frac{2 - 1.4}{2 - 1} 100 + \frac{1.4 - 1}{2 - 1} 105 = 102$$

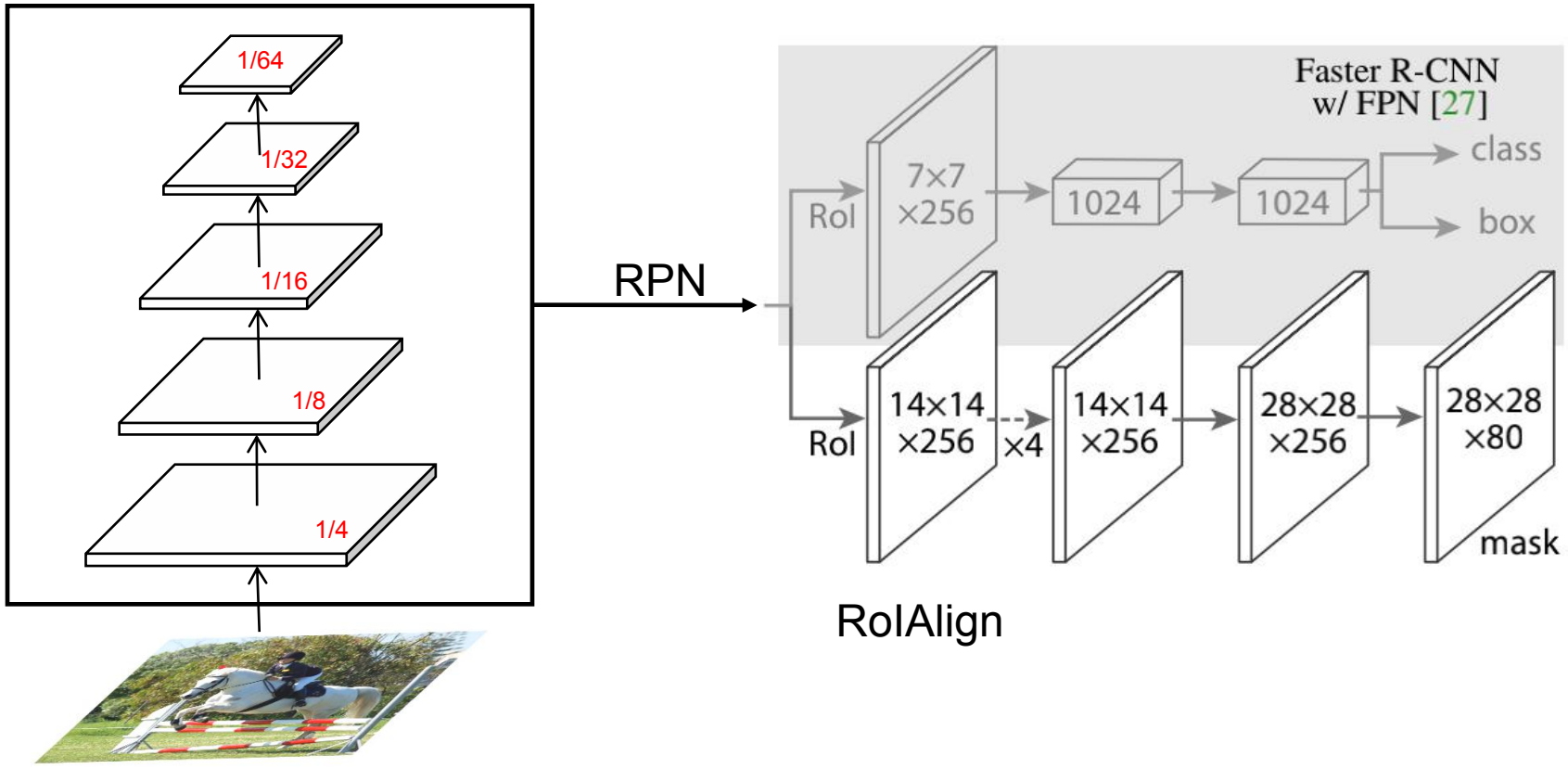
$$f(R_2) = \frac{x_2 - x}{x_2 - x1} Q_{12} + \frac{x - x_1}{x_2 - x_1} Q_{22} = \frac{2 - 1.4}{2 - 1} 110 + \frac{1.4 - 1}{2 - 1} 112 = 110.8$$

$$f(P) = \frac{y_2 - y}{y_2 - y_1} R_1 + \frac{y - y_1}{y_2 - y_1} R_2 = \frac{2 - 1.7}{2 - 1} 102 + \frac{1.7 - 1}{2 - 1} 110.8 = 108.16$$

最终计算得到P值为108.16, 那么在一个bin中4个值取最大即作为该bin的取值.

一、论文解读

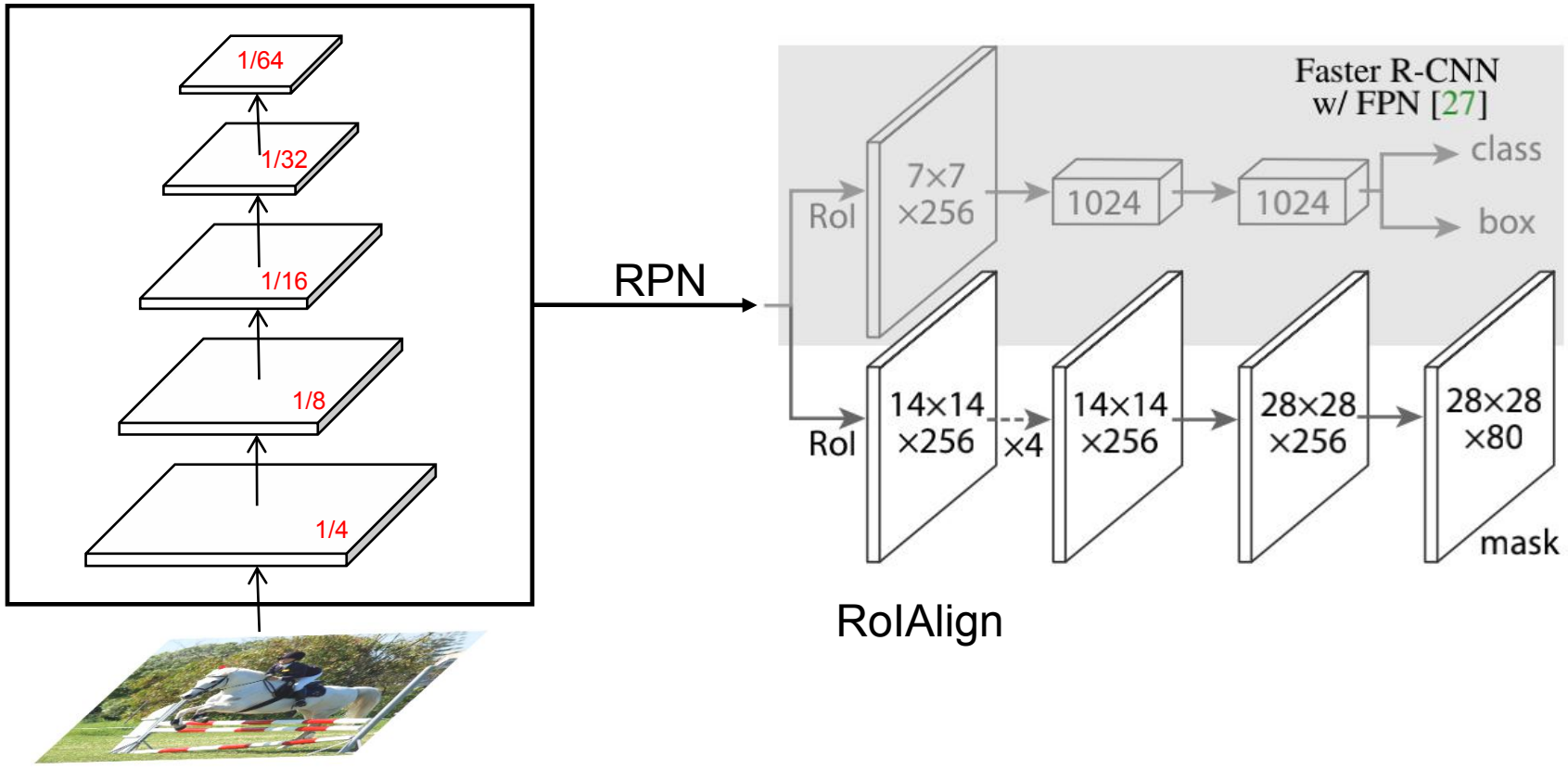
（四）、bbox, class, mask预测



class/box预测：rpn输出一系列roi， RoIAlign将roi逐个对应回Resnet输出的5个特征层中的一个，再对该特征做相应的裁剪，resize操作得到对应的特征（如图7x7x256），再对该特征做进一步卷尺,全连接最终输出预测.

一、论文解读

（四）、bbox, class, mask预测



mask预测：rpn输出一系列roi， RoIAlign将roi逐个对应回Resnet输出的5个特征层中的一个，再对该特征做相应的裁剪，resize操作得到对应的特征（如图14x14x256），再对该特征做进一步卷尺，上采样最终输出mask预测. mask预测输出统一为28x28大小，后面再根据预测的边框将mask的大小resize到匹配的大小。

二、开源代码解读

（一）、代码整体概况

inputs:

input_image, input_image_meta, input_rpn_match, input_rpn_bbox,
input_gt_class_ids, input_gt_boxes, input_gt_masks

model(training):

```
# 前卷积网络
C2, C3, C4, C5 = resnet_graph(*)
P2, P3, P4, P5, P6 = top_down_layer(*)

# rpn网络 roi筛选, 及其目标值计算
rpn_class_logits, rpn_class, rpn_bbox = rpn_graph(*)
anchors = get_anchors(*)
rpn_rois = ProposalLayer(*)
rpn_match, rpn_bbox = build_rpn_targets(*)

# 最终roi,scores,mask预测，及其目标值计算
rois, target_class_ids, target_bbox, target_mask = DetectionTargetLayer(*)
mrcnn_class_logits, mrcnn_class, mrcnn_bbox = fpn_classifier_graph(*)
mrcnn_mask = build_fpn_mask_graph(*)
```

二、开源代码解读

（一）、代码整体概况

inputs:

input_image, input_image_meta, input_anchors

model(predict):

```
# 前卷积网络
C2, C3, C4, C5 = resnet_graph(*)
P2, P3, P4, P5, P6 = top_down_layer(*)

# rpn网络
rpn_class_logits, rpn_class, rpn_bbox = rpn_graph(*)

# 最终roi,scores,mask预测
mrcnn_class_logits, mrcnn_class, mrcnn_bbox = fpn_classifier_graph(*)
mrcnn_mask = build_fpn_mask_graph(*)
detections = DetectionLayer(*)
```

二、开源代码解读

（二）、模型输入

模型数据的定义在data_generator方法中:

input_image: [batch, H, W, C]

input_image_meta:[batch, image_id, original_shape, resize_shape,
window, scale, active_class_ids]

input_rpn_match: [N]

input_rpn_bbox: [positive_nums, (dy, dx, log(dh), log(dw))]

input_gt_class_ids:[batch, instance_count]

input_gt_boxes:[batch, instance_count, (y1, x1, y2, x2)]

input_gt_masks:[batch, H, W, instance_count]

input_anchors: [num_anchors, (y1, x1, y2, x2)]

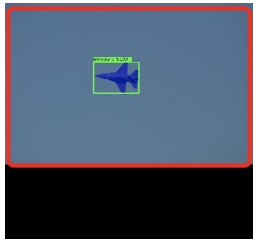
二、开源代码解读

（二）、模型输入

模型输入数据的细节：

1. input_image/mask 是经过resize和padding，全部统一了尺寸。

2. input_image_meta中的window是[y1, x1, y2, x2]，对应resize后的图片，由于最终统一尺寸，该window对应的是没有padding的区域，如下图红色框。



3. input_image_meta中scale表示缩放尺度。

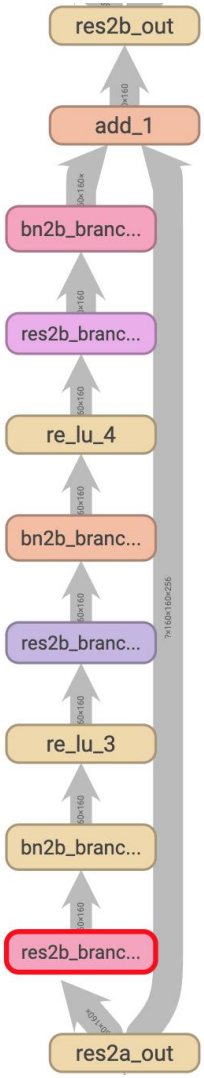
4. input_rpn_match和input_rpn_bbox是为了计算rpn损失的目标数据，match数据是每个生成的anchor对应的0/1背景前景类别，bbox是match=1那些bbox的平移缩放量.

二、开源代码解读

（三）、resnet,top_down layer: resnet两个模块

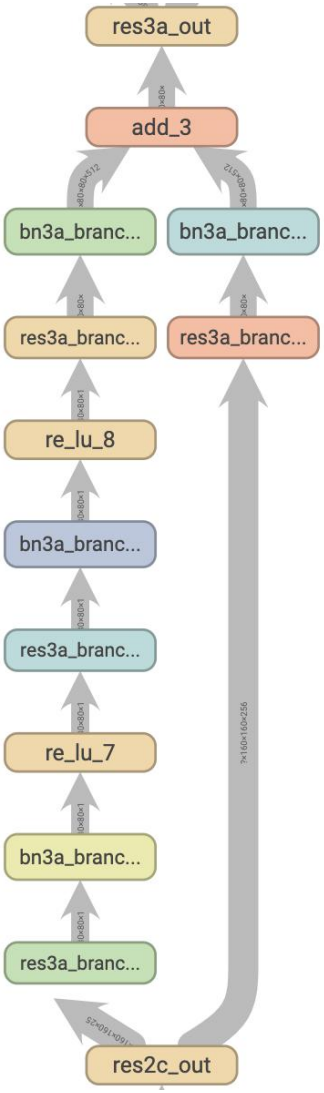
resnet: identity_block

卷积->bn->relu ->
卷积->bn->relu ->
卷积->bn->add ->
relu



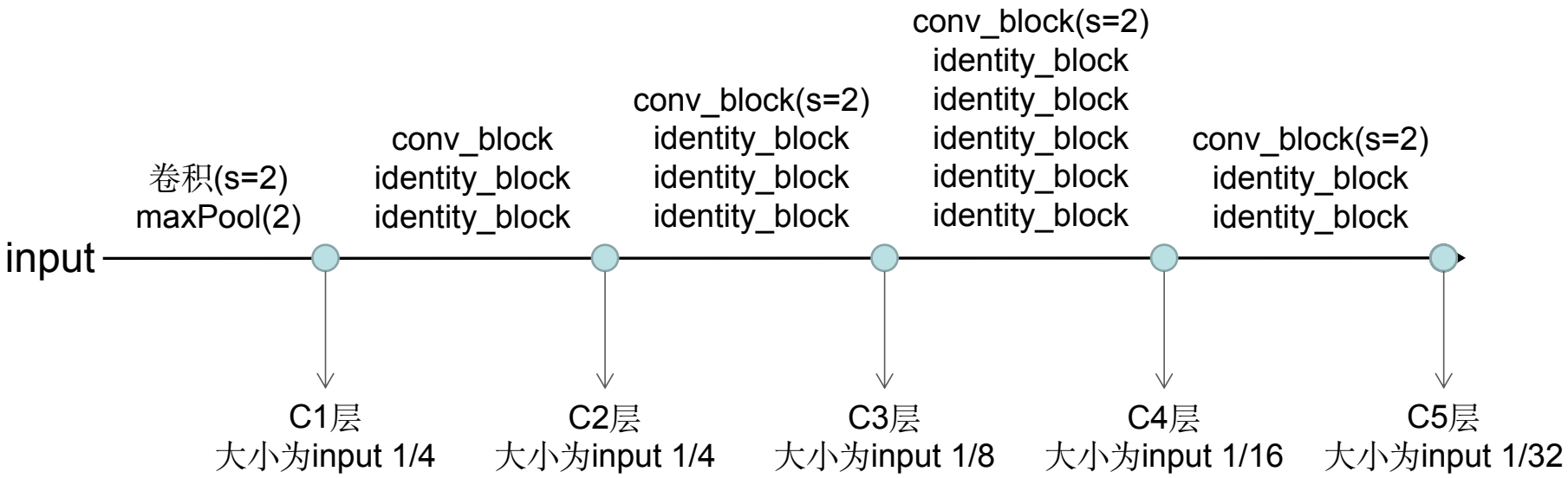
resnet: conv_block

卷积->bn->relu ->
卷积->bn->relu ->
卷积->bn->
add(卷积->bn) ->
relu



二、开源代码解读

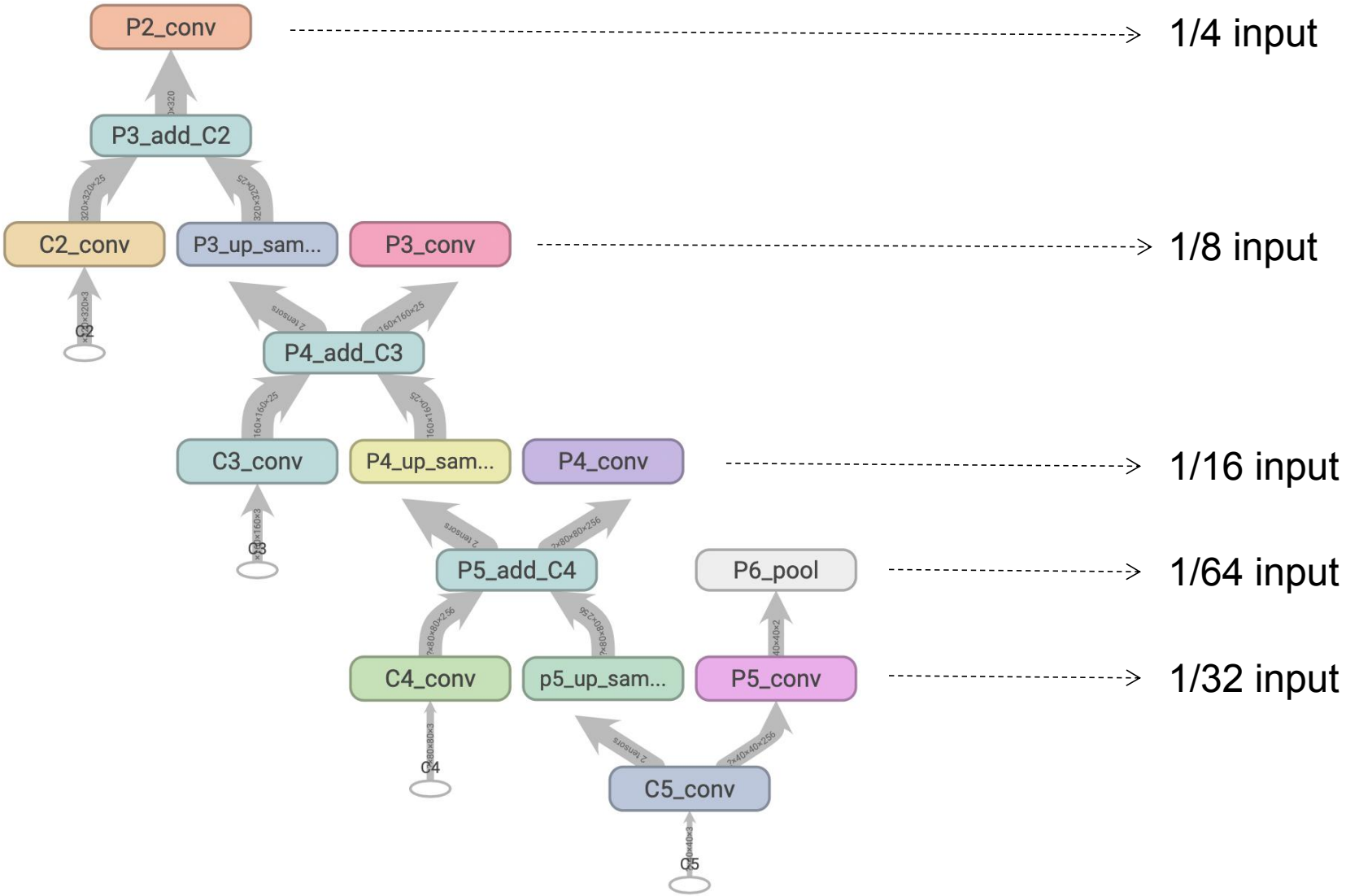
（三）、resnet,top_down layer: resnet整体结构及中间层输出



二、开源代码解读

（三）、resnet,top_down layer: top and down 最终特征层

最终5层输出shape:



二、开源代码解读

（四）、rpn及其target

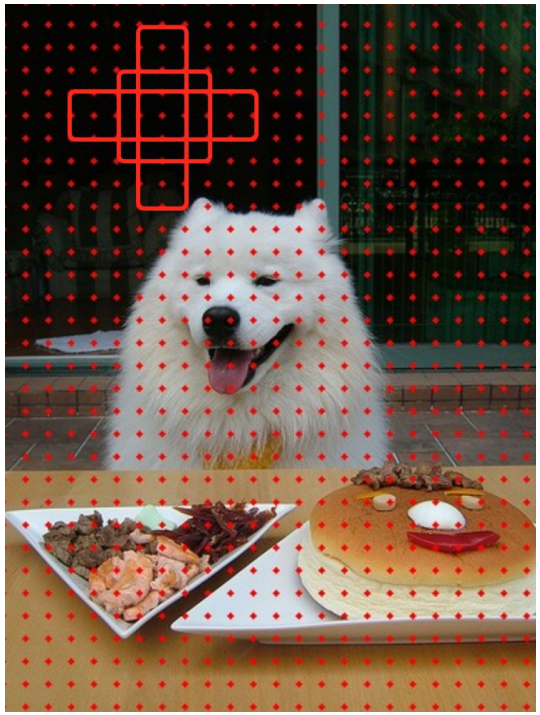
rpn网络两个输出, rpn_class, rpn_bbox:

```
shared = tf.keras.layers.Conv2D(512, (3, 3), padding='same')(feature_map)
rpn_class = tf.keras.layers.Conv2D(2 * anchors_per_location, (1, 1))(shared)
rpn_bbox = tf.keras.layers.Conv2D(4 * anchors_per_location, (1, 1))(shared)
```

anchor生成:

1. anchor以P2/3/4/5/6 共5层特征大小为基础，在每个特征点上生成长宽比1:1,1:2,2:1的三个边框(如右图红色边框)，这里与faster-rcnn不一样的是，faster-rcnn卷积输出的特征是单独一层，所以会在每个特征点上生成3个缩放比例与3种尺度大小的边框，共9个。

2. P2/3/4/5/6 5层特征对应的边框长分别为:
[32, 64, 128, 256, 512]，所以每层对应检测目标尺寸也有所不同，越下层检测目标越小，越上层检测目标越大。



二、开源代码解读

（四）、rpn及其target

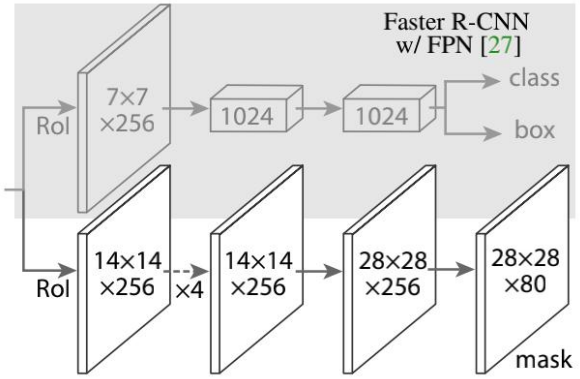
rpn taget计算:

1. build_rpn_targets:
   - 计算每个anchor与每个gt_box的overlap
   - overlap > 0.7 的标记为前景, < 0.3 的标记为背景, 其他没用
   - 计算每个anchor跟overlap最大的gt_box的平移缩放量

2. ProposalLayer:
   - 对前景预测概率取ton n个
   - 索引出对应的前景概率, 边框, 和anchor
   - 将索引出的边框平移缩放量映射到anchor作为输出的roi

二、开源代码解读

（五）、mask-rcnn及其target



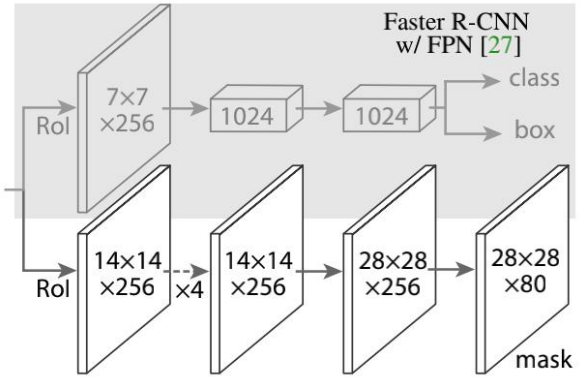
mask-rcnn box/class预测：

```
pooled = PyramidROIAlign([pool_size, pool_size])(rois, mrcnn_feature_maps)
⋮
mrcnn_class_logits = tf.keras.layers.TimeDistributed(
                                tf.keras.layers.Dense(self.num_classes)
                            )(shared)

mrcnn_bbox = tf.keras.layers.TimeDistributed(
                        tf.keras.layers.Dense(self.num_classes * 4, activation='linear')
                           )(shared)
```

二、开源代码解读

（五）、mask-rcnn及其target



mask-rcnn mask预测：

```
pooled = PyramidROIAlign([pool_size, pool_size])(rois, mrcnn_feature_maps)
⋮
x = tf.keras.layers.TimeDistributed(
        tf.keras.layers.Conv2DTranspose(filters=256,
                                        kernel_size=(2, 2),
                                        strides=2)
    )(x)
mrcnn_mask = tf.keras.layers.TimeDistributed(
                    tf.keras.layers.Conv2D(filters=self.num_classes,
                                           kernel_size=(1, 1),
                                           strides=1,
                                           activation="sigmoid"),
                )(x)
```

二、开源代码解读

（五）、mask-rcnn及其target

mask-rcnn DetectionTargetLayer计算：

1. 计算每个roi与每个gt_box的overlap
2. overlap > 0.5 的标记为前景，以最大的作为前景类别，< 0.5 的标记为背景
3. 计算每个anchor跟overlap最大的gt_box的平移缩放量，作为box目标值
4. 将前景roi对gt_mask数据进行裁剪，resize到固定大小，作为mask目标值

5. 最后将class,box,mask多个batch数据 padding 到相同大小

二、开源代码解读

（六）、loss

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

L_{cls} 类别损失: rpn_class和mrcnn_class的类别损失都是交叉熵损失

L_{box} 边框损失:

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

其中rpn_box只对rpn_target_class = 1的数据计算
mrcnn_box只对mrcnn_target_class > 0的数据计算

L_{mask} 遮罩掩码损失: 只对mrcnn_target_class下的mrcnn_mask计算0/1交叉熵损失

二、源码训练

- Pascal voc2012分割数据color map:

```
[0 0 0]
[128   0   0]
[  0 128   0]
[128 128   0]
[  0   0 128]
[128   0 128]
[  0 128 128]
[128 128 128]
[64  0  0]
[192   0   0]
[ 64 128   0]
[192 128   0]
[ 64   0 128]
[192   0 128]
[ 64 128 128]
[192 128 128]
[ 0 64  0]
[128  64   0]
[  0 192   0]
[128 192   0]
[  0  64 128]
```