Mask\_rcnn技术文档

# 概述：

## 资料地址：

### 论文地址

<https://arxiv.org/abs/1703.06870> （Kaiming He何恺明,2017）代表作品：

Dark Channel去雾、Faster R-CNN、ResNet、Mask R-CNN。

### 代码地址

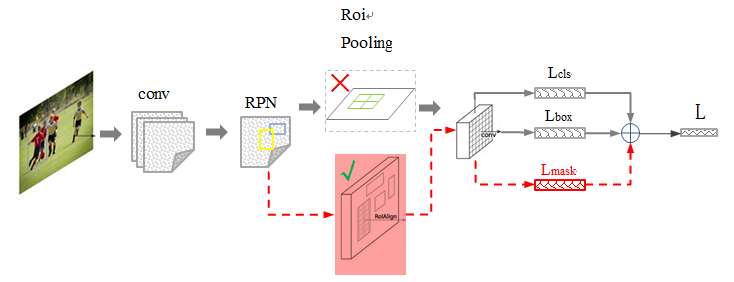
<https://github.com/facebookresearch/Detectron>

### 镜像地址

192.168.15.100：/home/tar/detectron\_gpu.tar

## 技术理论

### Mask\_rcnn网络结构及特点



  其中黑色部分为原来的 Faster-RCNN，红色部分为在 Faster网络上的修改。

mask rcnn主要的贡献在于如下：

**●** **强化的基础网络**

通过 ResNeXt-101+FPN 用作特征提取网络，达到 state-of-the-art 的效

果。 FPN（feature pyramid networks）

**●** **将 Roi Pooling 层替换成了 RoiAlign，**解决Misalignment 的问题。

**●** **Loss Function：**

由原来的 FCIS 的 **基于单像素softmax的多项式交叉熵**变为了**基于单像素sigmod二值交叉熵**， softmax会产生FCIS的 ROI inside map与ROI outside map的竞争。但文章作者确实写到了类间的竞争，二值交叉熵会使得每一类的 mask 不相互竞争，而不是和其他类别的 mask 比较 。

Fully Convolutional Instance-aware Semantic Segmentation (FCIS) 2016年

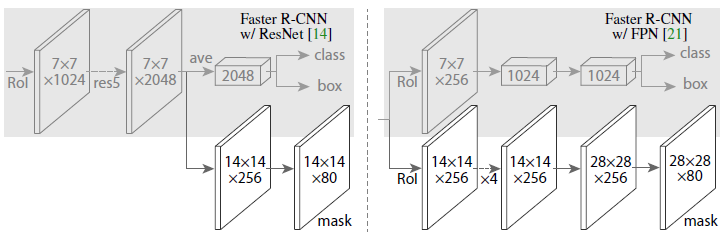
文章还指出在物体检测中，Mask-RCNN 比一般的 state-of-the-art 方法（用Fatser-RCNN+ResNet-101+FPN实现）在 bounding-box 的 mAP 上高3.6个点，1.1来自ROIAlign，0.9来自多任务训练，1.6来自更好的基础网络（ResNeXt-101）

**参考文献**： <https://www.zhihu.com/question/57403701>

### Mask-RCNN 技术要点

**● 技术要点1 - 强化的基础网络**

  通过 ResNeXt-101+FPN 用作特征提取网络，达到 state-of-the-art 的效果。左图没有FPN，右图结构有FPN。



**备注：**是81类（14\*14\*81,80应该是没加背景）

**● 技术要点2 - ROIAlign**

    采用 ROIAlign 替代 RoiPooling（改进池化操作）。引入了一个插值过程，先通过双线性插值将 7\*7插值到14\*14，再 pooling到7\*7，很大程度上解决了仅通过 Pooling 直接采样带来的 Misalignment 对齐问题。

  虽然在分类问题上影响并不大，但在 Pixel 级别的 Mask 上会存在较大误差。能够看到 ROIAlign 带来较大的改进，可以看到，Stride 越大改进越明显。

**● 技术要点3 - Loss Function**

最终的loss为分类，回归以及分割之和。

这里写图片描述

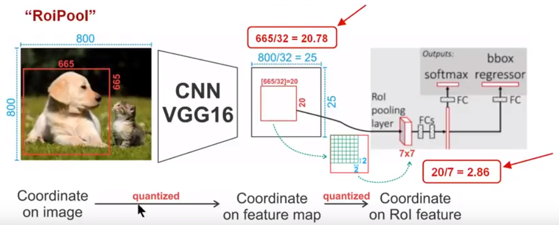
   每个 ROIAlign 对应 K \* m^2 维度的输出。**K 对应类别个数，即输出 K 个mask**，m对应 池化分辨率（7\*7）。Loss 函数定义：

**Lmask(Cls\_k) = Sigmoid (Cls\_k)，    平均二值交叉熵 （average binary cross-entropy）Loss，通过逐像素的 Sigmoid 计算得到。**

   通过对每个 Class 对应一个 Mask 可以有效避免类间竞争（其他 Class 不贡献 Loss，意思是loss只计算预测类别的loss ）。

### ROI Align 的基本原理

**● ROI Pooling 的局限性分析（两次量化）**



这是一个Faster-RCNN检测框架。输入一张800\*800的图片，图片上有一个665\*665的包围框(框着一只狗)。图片经过主干网络提取特征后，特征图缩放步长（stride）为32。因此，图像和包围框的边长都是输入时的1/32。800正好可以被32整除变为25。但665除以32以后得到20.78，带有小数，于是ROI Pooling **直接将它量化成20**。接下来需要把框内的特征池化成7\*7的大小，因此将上述包围框平均分割成7\*7个矩形区域。显然，每个矩形区域的边长为2.86（20/7=2.86），又含有小数。于是ROI Pooling **再次把它量化到2**。经过这两次量化，候选区域已经出现了较明显的偏差（如图中绿色部分所示）。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。**那么0.8的偏差，在原图上就是接近30个像素点的差别**，这一差别不容小觑。

**● ROI** **Align 的主要思想和具体方法**

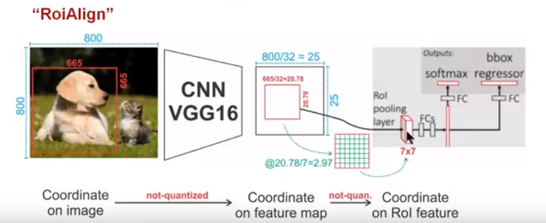


图2

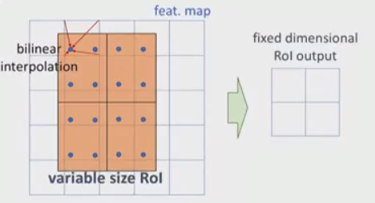


图3

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(如图2)。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，如图3所示：

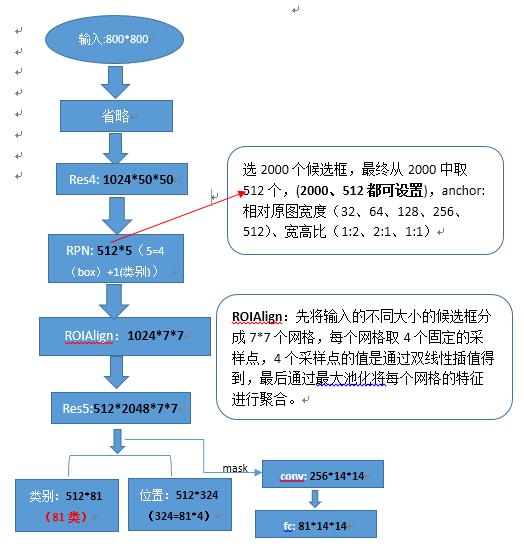
* 遍历每一个候选区域，保持浮点数边界不做量化。
* 将候选区域分割成k x k个单元，每个单元的边界也不做量化。
* 在每个单元中计算**固定四个坐标位置**，用双线性内插的方法计算出这四个位置的值，然后进行最大池化操作。

（图3的解释：将候选区域分成2\*2个网格，每个网格取4个采样点，根据双线性插值计算4个采样点的值，再通过最大池化将每个网格的特征进行聚合，输出2\*2大小的ROI）

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为中心点。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ROIPooling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）。

**参考文献：** <http://blog.leanote.com/post/afanti.deng@gmail.com/b5f4f526490b>

### 整体流程

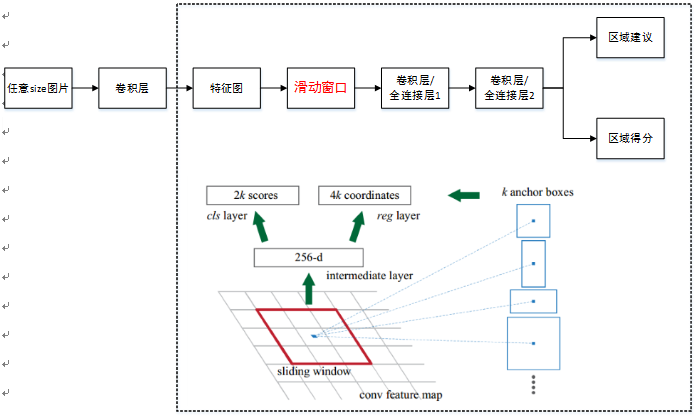


## 3.其他

### 1）RPN（区域生成网络）

RPN是regional proposal networks的缩写，是faster-RCNN（ [Kaiming He](https://arxiv.org/find/cs/1/au:+He_K/0/1/0/all/0/1)，2015）结构中的一部分。利用CNN卷积操作后的特征图生成region proposals，代替了Selective Search、EdgeBoxes等方法，速度上提升明显。

单个RPN网络结构如下图：



**解释：**假如给定600\*1000的输入图像，经过卷积操作得到最后一层的卷积feature map(大小为40\*60)，最后一层卷积层共有256个feature map。

在这个特征图上使用3\*3的卷积核（**滑动窗口**）与特征图进行卷积，那么这个3\*3的区域卷积后可以获得一个256维的特征向量。因为这个3\*3的区域上，每一个特征图上得到一个1维向量，256个特性图即可得到256维特征向量。 （k=3, p=1,s=1, 输出=（输入-k+2\*p）/s +1）

3\*3滑窗中心点位置，对应预测输入图像3种尺度（128,256,512），3种长宽比（1:1,1:2,2:1）的regionproposal，这种映射的机制称为**anchor**，产生了k=9个anchor。即每个3\*3区域可以产生9个region proposal。所以对于这个40\*60的feature map，总共有约20000(40\*60\*9)个anchor，也就是预测20000个region proposal。

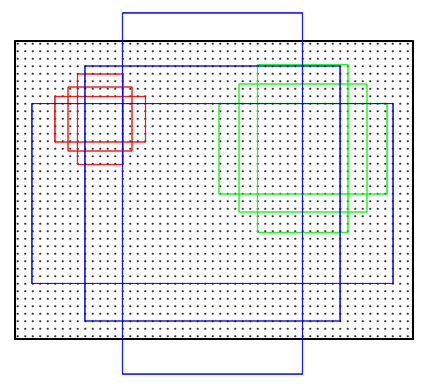
**分类层（cls\_score）**输出每一个位置上，9个anchor属于前景和背景的概率；

**窗口回归层（bbox\_pred）**输出每一个位置上，9个anchor对应窗口应该平移缩放的参数。   
对于每一个位置来说，分类层从256维特征中输出属于前景和背景的概率；窗口回归层从256维特征中输出4个平移缩放参数。

**作用：**供RPN网络输入的特征图经过RPN网络得到区域建议和区域得分，并对区域得分采用非极大值抑制【阈值为0.7】，输出其Top-N【文中为300】得分的区域建议给RoI池化层；

#### 候选区域（anchor）

特征可以看做一个尺度40\*60的256通道图像，对于该图像的每一个位置，考虑9个可能的候选窗口：**三种面积{128\*128,256\*256,512\*512}×三种比例{1:1,1:2,2:1}**。这些候选窗口称为**anchors**。下图示出51\*39个anchor中心，以及9种anchor示例。(宽高比=**1:2 = 128/sqrt(2) : 128\*sqrt(2), 宽度128指的是在原图上的大小，对于特征图40\*60上anchor的宽度=128\*40/原图宽度** )



作者在文中表示采用**Anchors这种方法具有平移不变性**，就是说在图像中平移了物体，窗口建议也会跟着平移。同时这种方式也减少了整个模型的size，输出层256×(4+2)×9=1.4×10^4个参数【256是前一层特征维度，(4+2)×9是9个Anchors的前景背景得分和平移缩放参数】

**问题：如何处理多尺度多长宽比问题？即如何使24×24和1080×720的车辆同时在一个训练好的网络中都能正确识别？**

作者提出了一种叫Anchors金字塔的方法来解决多尺度多长宽比的问题，在RPN网络中对特征图滑窗时，对滑窗位置中心进行多尺度多长宽比的采样，并对多尺度多长宽比的anchor boxes区域进行回归和分类，利用**Anchors金字塔**就仅仅依赖于单一尺度的图像和特征图和单一大小的卷积核，就可以解决多尺度多长宽比问题，这种对推荐区域采样的模型不管是速度还是准确率都能取得很好的性能。

**问题：同传统滑窗方法提取区域建议方法相比，RPN网络有什么优势？**  
传统方法是训练一个能检测物体的网络，然后对整张图片进行滑窗判断，由于无法判断区域建议的尺度和长宽比，所以需要**多次缩放**，这样找出一张图片有物体的区域就会很慢；   
虽然RPN网络也是用滑动窗口策略，但是滑动窗口是在卷积层特征图上进行的，维度较原始图像降低了很多倍【中间进行了多次max pooling 操作】,RPN采取了9种不同尺度不同长宽比的anchors，同时最后进行了bounding-box回归，即使是这9种anchors外的区域也能得到一个跟目标比较接近的区域建议。

#### 区域生成网络：训练

##### 样本

考察训练集中的每张图像：   
a. 对**每个标定**的真值候选区域，与其重叠比例最大的anchor记为**前景样本**   
b. 对a)剩余的anchor，如果其**与某个标定**重叠比例大于0.7，记为**前景样本**；如果其**与任意**一个标定的重叠比例都小于0.3，记为**背景样本**   
c. 对a),b)剩余的anchor，弃去不用。   
d. 跨越图像边界的anchor弃去不用

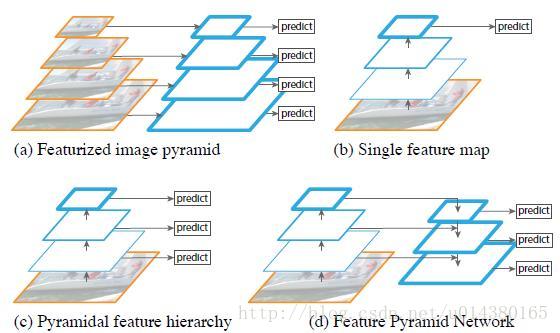
**参考文献**： <http://blog.csdn.net/wopawn/article/details/52223282>

<http://blog.csdn.net/shenxiaolu1984/article/details/51152614>

### 2）FPN （feature pyramid networks）

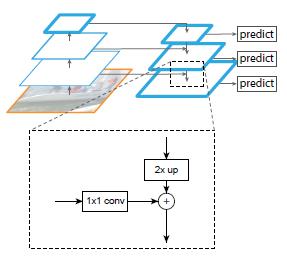
《feature pyramid networks for object detection 》**这篇论文是CVPR2017年的文章，采用特征金字塔做目标检测。**

原来多数的object detection算法都是只采用顶层特征做预测，但我们知道**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。**



（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。这种方法的缺点在于增加了时间成本。有些算法会在测试时候采用图像金字塔。   
（b）像SPP net，Fast RCNN，Faster RCNN是采用这种方式，即仅采用网络**最后一层的特征进行预测**。   
（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。**

**作者的主网络采用ResNet。**   
作者的算法大致结构如下Fig3：**一个自底向上的线路，一个自顶向下的线路，横向连接（lateral connection，对应位置相加）**图中放大的区域就是横向连接，这里**1\*1的卷积核的主要作用**是减少卷积核的个数，也就是**减少了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。在前向过程中，feature map的大小在经过某些层后会改变，而在经过其他一些层的时候不会改变，作者将不改变feature map大小的层归为一个stage，因此每次抽取的特征都是每个stage的最后一个层输出，这样就能构成特征金字塔。   
**自顶向下**的过程采用上采样（upsampling）进行，而**横向连接**则是将**上采样的结果和自底向上生成的相同大小的feature map进行融合（merge）**。在融合之后还会再采用3\*3的卷积核对每个融合结果进行卷积，**目的是消除上采样的混叠效应**（aliasing effect）。并假设生成的feature map结果是P2，P3，P4，P5，和原来自底向上的卷积结果C2，C3，C4，C5一一对应。

**问题：不同深度的 feature map 为什么可以经过 upsample 后直接相加？**

A：作者解释说这个原因在于我们做了 end-to-end 的 training，因为不同层的参数不是固定的，不同层同时给监督做 end-to-end training，所以相加训练出来的东西能够更有效地融合浅层和深层的信息。

**参考文献**：<http://blog.csdn.net/u014380165/article/details/72890275>

# 代码调试

## 运行工程

### 测试

1. 指定文件夹测试：

python2 tools/infer\_simple.py \

--cfg configs/12\_2017\_baselines/e2e\_mask\_rcnn\_R-101-FPN\_2x.yaml \

--output-dir /tmp/detectron-visualizations \

--image-ext jpg \

--wts https://s3-us-west-2.amazonaws.com/detectron/35861858/12\_2017\_baselines/e2e\_mask\_rcnn\_R-101-FPN\_2x.yaml.02\_32\_51.SgT4y1cO/output/train/coco\_2014\_train:coco\_2014\_valminusminival/generalized\_rcnn/model\_final.pkl \

demo

参数说明：--cfg 配置文件，最终的配置文件为/lib/core/config.py，--output-dir 输出目录，--wts 模型，demo 测试文件

1. 使用配置文件中的数据集测试：

python2 tools/test\_net.py \

--cfg configs/12\_2017\_baselines/e2e\_mask\_rcnn\_R-101-FPN\_2x.yaml \

TEST.WEIGHTS https://s3-us-west-2.amazonaws.com/detectron/35861858/12\_2017\_baselines/e2e\_mask\_rcnn\_R-101-FPN\_2x.yaml.02\_32\_51.SgT4y1cO/output/train/coco\_2014\_train:coco\_2014\_valminusminival/generalized\_rcnn/model\_final.pkl \

NUM\_GPUS 1

参数说明：--cfg 配置文件，TEST.WEIGHTS 模型，测试文件使用配置文件中的测试集

### 训练

1、使用单个GPU

python2 tools/train\_net.py \

--cfg configs/12\_2017\_baselines/e2e\_mask\_rcnn\_R-101-FPN\_2x.yaml \

OUTPUT\_DIR /tmp/detectron-output

2、使用多个GPU

python2 tools/train\_net.py \

--multi-gpu-testing \

--cfg configs/12\_2017\_baselines/e2e\_mask\_rcnn\_R-101-FPN\_2x.yaml \

OUTPUT\_DIR /tmp/detectron-output

## 测试调试

### 调用关系

Text\_net.py -> child\_func = test\_net -> test\_engine.py -> test\_net() -> im\_detect\_all() -> core/test.py -> im\_detect\_all()。

### 检测函数im\_detect\_all()详解

1. score,box,im\_scalcs = im\_detect\_bbox(model,im,)

入参：model -- 模型；

im -- 图片。

出参：scores -- （1000 ， 81），每一类检测出1000个候选框的得分；

boxes -- （1000 ， 81\*4），每一类检测出1000个候选框的坐标；

im\_scales -- 图片进入网络缩放尺度。

功能：每一类物体检测出1000个样本。

1. scores，boxes，cls\_boxes = box\_results\_nms\_and\_limit(scores,boxes)

入参：scores -- （1000 ， 81），每一类检测出1000个候选框的得分；

boxes -- （1000 ， 81\*4），每一类检测出1000个候选框的坐标。

出参：scores -- （62 ，1），筛选后62个候选框的得分；

boxes -- （62 , 4），筛选后62个候选框的坐标，并且回归到原图尺寸；

cls\_boxes -- 最终输出的bbox和其对应score。

0(背景）：[]

1(人)：[box(1\*4),score(1)], [box(1\*4),score(1)]

2(物体1)：[]

cls\_boxes(list\_len81) 3(物体2)：[box(1\*4),score(1)]

.

.

.

80(物体n)：[]

功能：对box进行筛选，首先对scores>0.5，nms<0.3，最后根据参数**cfg.TEST.DETECTIONS\_PER\_IM=100,每一类取分值最高的100个**。

1. masks = im\_detect\_mask(model,im\_scales,boxes)

入参：model -- 模型；

im\_scales -- 图片进入网络缩放尺度；

boxes -- （62 , 4），筛选后62个候选框的坐标。

出参：masks -- （62,81,28,28），62个候选框对应的分割数据，每一个候选框对每一类进行分割；

功能：根据检测出的物体个数进行分割提取。

1. cls\_segms = segm\_results(cls\_boxes,masks,boxes,im\_shape[0],im\_shape[1])

入参：cls\_boxes -- 最终输出的bbox和其对应score；

masks -- （62,81,28,28），62个候选框对应的分割数据，每一个候选框对每一类进行分割；

boxes -- （62 , 4），筛选后62个候选框的坐标。

im\_shape[0] -- 输入图片的高；

im\_shape[1] -- 输入图片的宽；

出参：cls\_segms -- 最终输出的分割信息，每个单元为原图大小的二值化图（0,1）分布。

功能：将每一个mask（28\*28）扩充到（30\*30），并根据boxes扩充到对用的坐标上，然后进行二值化处理，最终生成一副原图大小的二值化图，这就为每一个mask的最终输出，压进存放分割信息的cls\_segms列表中。

## 训练调试

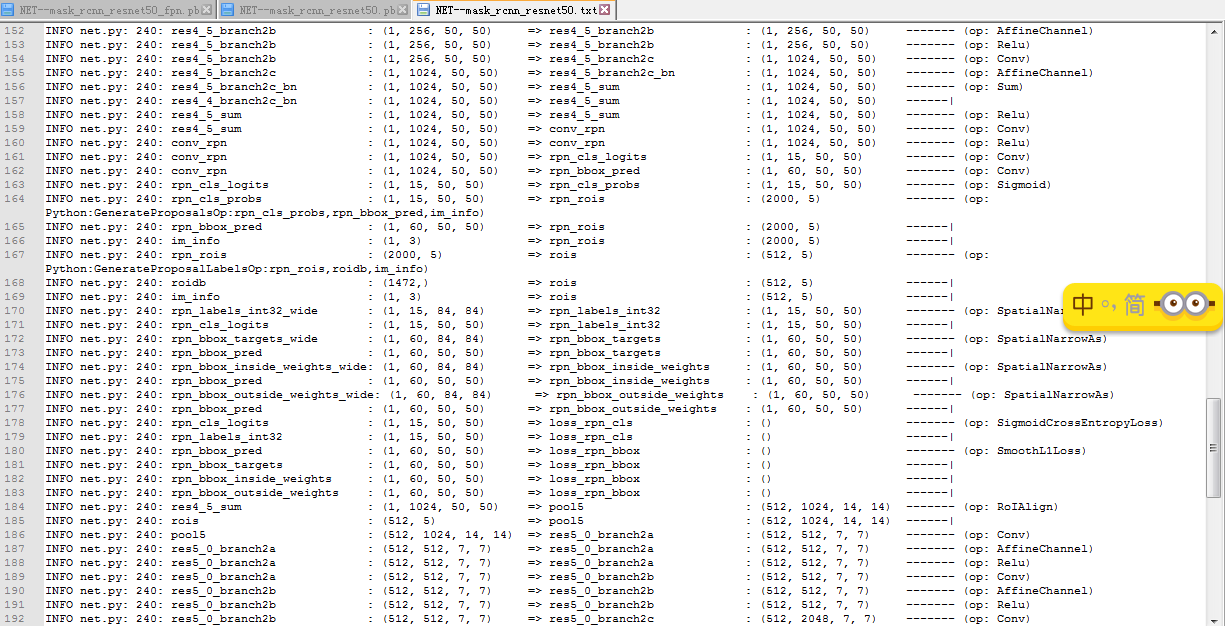
### 调用关系

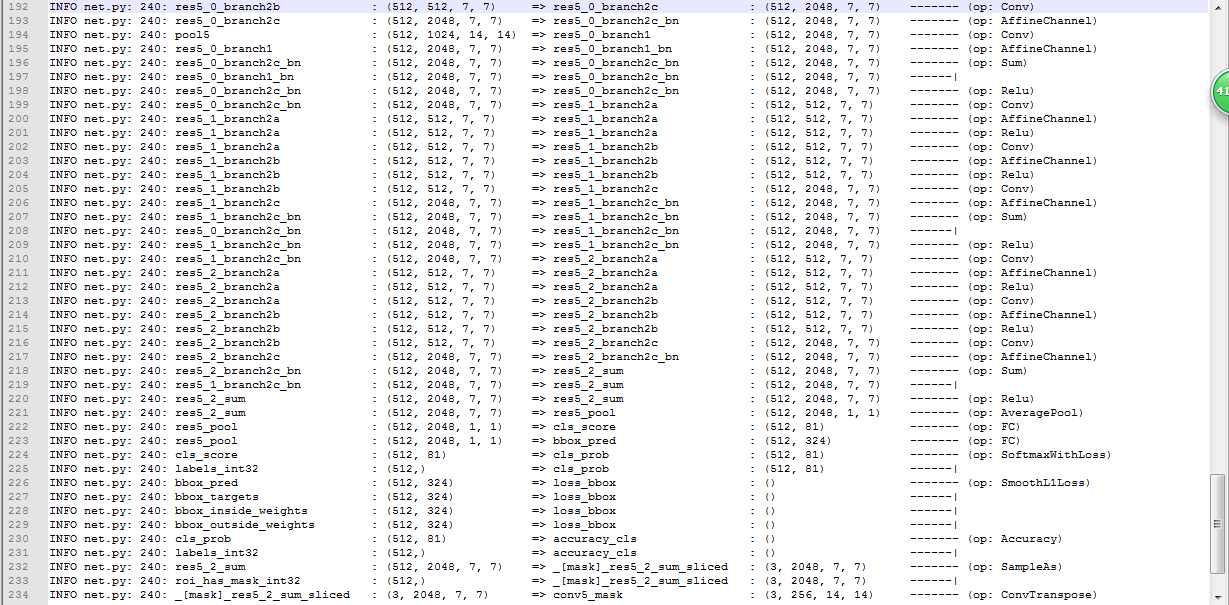
Train\_net.py -> main() -> train\_model() -> creat\_model() -> model\_builder.creat(cfg.MODEL.TYPE,train=True) -> modle\_builder.py -> create(cfg.MODEL.TYPE,train=True) -> generalized\_rcnn(model)P83 -> build\_generic\_detection\_model().

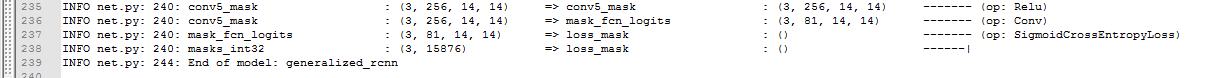
### 网络示例

下图为mask\_rcnn网络训练示例：

使用resnet50，并且没有加入fpn，下图省略resnet50前4层的网络，重点输出了rpn以及mask的网络。







# 训练body-正面、侧面、背面调试过程

## 准备工作

### 数据标注

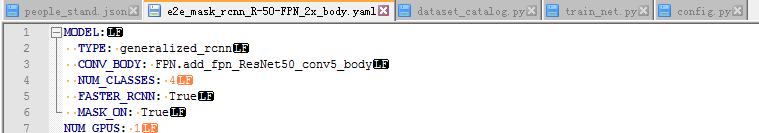
使用labelme工具进行标注，进行分割标注。

### 生成json文件

使用脚本/home/ligang/labelme/labelme\_to\_labelcoco.py生成json文件。

### 修改配置文件

创建训练配置文件Detectron-master/configs/12\_2017\_baselines /e2e\_mask\_rcnn\_R-50-FPN\_2x\_body.yaml。将e2e\_mask\_rcnn\_R-50-FPN\_2x.yaml复制到上述文件。修改内容如下：



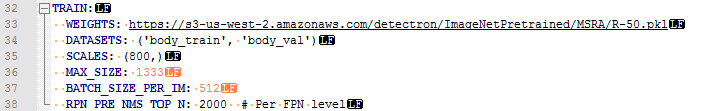
NUM\_CLASS：4

标注的标签为：front、side、back和背景；

NUM GPUS：1

本环境在100服务器上，使用1块GPU。

此处设置，先查看测试的参数修改



DATASETS：(‘body\_train’,’body\_val’)

此处的body\_train和body\_val索引在文件Detectron-master /lib/datasets dataset\_catalog.py中使用，增加这两个数据集的图片以及json文件路径，修改方法如下图：



将训练的图片放入Detectron-master/lib/datasets/data /body/body\_train，

将测试的图片放入Detectron-master/lib/datasets/data /body/body\_val，

将训练集的instances\_body\_train.json放入/body/annotations，

将测试集的instances\_body\_val.json放入/body/annotations。

## 训练

首先 进入 Detectron-master/lib路径下：

make

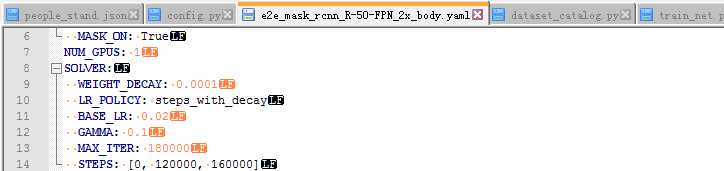
这样避免本地代码和镜像的代码版本不匹配时报错！！！

python2 tools/train\_net.py \

--cfg configs/getting\_started/e2e\_mask\_rcnn\_R-50-FPN\_2x\_body.yaml \

OUTPUT\_DIR /opt/ligang/Detectron-master/detectron-output

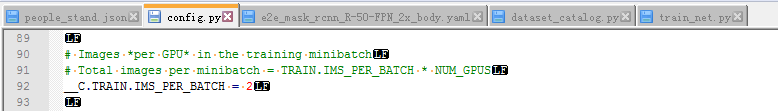
### 修改参数



e2e\_mask\_rcnn\_R-50-FPN\_2x\_body.yaml -> SOLVER -> BASE\_LR: 0.02

修改为：BASE\_LR: 0.001

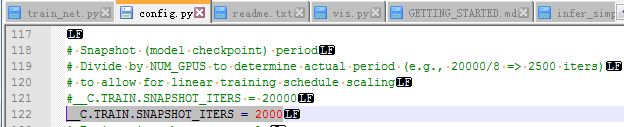
功能：学习率，默认0.02有可能训练会跑飞。



Detectron-master/lib/core/config.py -> \_\_C.TRAIN.IMS\_PER\_BATCH = 2

修改为：设置为1，以实际情况修改，看GPU内存使用率

功能：batchsize，作者默认为2，其中用8块GPU训练，所以为8\*2 = 16；



Detectron-master/lib/core/config.py -> \_\_C.TRAIN.SNAPSHOT\_ITERS = 20000 -> \_\_C.TRAIN.SNAPSHOT\_ITERS = 2000

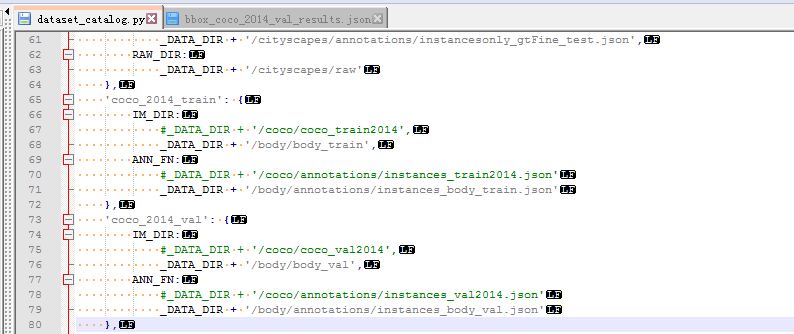
功能：模型保存

## 测试

python2 tools/test\_net.py --cfg configs/12\_2017\_baselines/e2e\_mask\_rcnn\_R-50-FPN\_2x\_body.yaml TEST.WEIGHTS /opt/ligang/Detectron-master/detectron-output/train/coco\_2014\_train:coco\_2014\_val/generalized\_rcnn/model\_iter889999.pkl NUM\_GPUS 1

### 修改参数

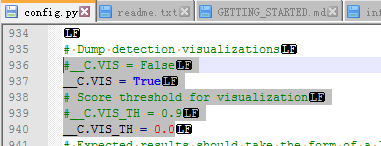
注：e2e\_mask\_rcnn\_R-50-FPN\_2x\_body.yaml中的TRAIN-> DATASETS 和 TEST -> DATASETS不能自己增加索引，否则在测试的时候不能正常计算AP值。



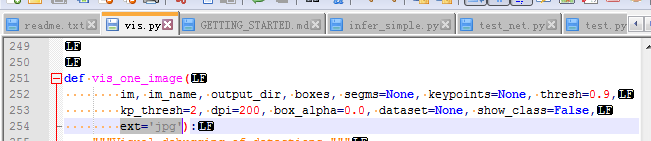
这里将 coco\_2014\_train和coco\_2014\_val的路径改为本次数据集的路径。

Detectron-master/lib/core/config.py -> \_\_C.VIS = False -> \_\_C.VIS = True; 控制测试时图片是否输出；

Detectron-master/lib/core/config.py -> \_\_C.VIS\_TH = 0.9 -> \_\_C.VIS\_TH = 0.0; 控制测试时，输出的图片的置信度大小；



/Detectron-master/lib/utils/vis.py -> vis\_one\_image(ext='pdf') -> vis\_one\_image(ext='jpg'); 控制测试时图片的输出格式。



# 测试结果统计

## 测试结果1

配置文件：--cfg configs/12\_2017\_baselines/e2e\_mask\_rcnn\_R-50-FPN\_2x\_body.yaml

训练集样本个数：front：553(48%)，side：434(38%)，back：164(14%)

测试集样本个数：front：157(52%)，side：103(34%)，back：40(13%)

bbox没有扩展

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 迭代次数 | Task | AP（IoU=[0.50,0.95]） | AP50 | AP75 | APs | APm | APl |
| 20000 | box | 0.8422 | 0.9951 | 0.9555 | 0.8 | 0.8376 | 0.8450 |
| mask | 0.8347 | 0.9951 | 0.9808 | 0.8 | 0.722 | 0.8515 |
| 30000 | box | 0.8727 | 0.9961 | 0.9903 | 0.9 | 0.8503 | 0.8789 |
| mask | 0.8551 | 0.9961 | 0.9798 | 0.9 | 0.7964 | 0.8755 |
| 50000 | box | 0.9189 | 0.9952 | 0.9952 | 0.8 | 0.8971 | 0.9252 |
| mask | 0.8754 | 0.9952 | 0.9952 | 0.6 | 0.8276 | 0.8884 |
| 75999 | box | 0.9370 | 0.9952 | 0.9952 | 0.9 | 0.9185 | 0.9408 |
| mask | 0.8913 | 0.9952 | 0.9952 | 0.8 | 0.8532 | 0.9000 |
| 100000 | box | 0.9616 | 0.9959 | 0.9959 | 1 | 0.9479 | 0.9667 |
| mask | 0.8976 | 0.9959 | 0.9959 | 0.9 | 0.9517 | 0.9086 |

bbox扩展原框的5%

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 迭代次数 | Task | AP（IoU=[0.50,0.95]） | AP50 | AP75 | APs | APm | APl |
| 20000 | box | 0.8582 | 0.9962 | 0.9947 | 0.6 | 0.8356 | 0.8651 |
| mask | 0.8406 | 0.9962 | 0.9792 | 0.8 | 0.7607 | 0.8626 |
| 30000 | box | 0.8787 | 0.9944 | 0.9944 | 0.9 | 0.8658 | 0.8842 |
| mask | 0.8470 | 0.9944 | 0.9944 | 0.8 | 0.7858 | 0.8661 |
| 50000 | box | 0.9054 | 0.9949 | 0.9949 | 1 | 0.9012 | 0.9092 |
| mask | 0.8750 | 0.9949 | 0.9949 | 0.9 | 0.8198 | 0.8894 |
| 75999 | box | 0.9339 | 0.9948 | 0.9948 | 1 | 0.9315 | 0.9364 |
| mask | 0.8866 | 0.9948 | 0.9948 | 0.9 | 0.8431 | 0.8998 |
| 100000 | Box | 0.9421 | 0.9955 | 0.9955 | 1 | 0.9416 | 0.9440 |
| mask | 0.8893 | 0.9955 | 0.9955 | 0.9 | 0.8549 | 0.8973 |

bbox扩展原框的10%

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 迭代次数 | Task | AP（IoU=[0.50,0.95]） | AP50 | AP75 | APs | APm | APl |
| 20000 | box | 0.8068 | 0.9940 | 0.9767 | 0.6 | 0.7978 | 0.8105 |
| mask | 0.8154 | 0.9940 | 0.9582 | 0.8 | 0.7369 | 0.8408 |
| 30000 | box | 0.8763 | 0.9953 | 0.9953 | 0.7 | 0.8773 | 0.8786 |
| mask | 0.8508 | 0.9953 | 0.9953 | 0.8 | 0.7883 | 0.8702 |
| 50000 | box | 0.9150 | 0.9947 | 0.9947 | 0.8 | 0.9050 | 0.9206 |
| mask | 0.8655 | 0.9947 | 0.9947 | 0.9 | 0.8046 | 0.8843 |
| 75999 | box | 0.9248 | 0.9947 | 0.9947 | 0.8 | 0.8934 | 0.9353 |
| mask | 0.8780 | 0.9947 | 0.9947 | 0.8 | 0.8212 | 0.8944 |
| 100000 | box | 0.9510 | 0.9945 | 0.9945 | 1 | 0.9316 | 0.9583 |
| mask | 0.8871 | 0.9945 | 0.9945 | 0.9 | 0.8475 | 0.8977 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 迭代次数 | Task | bbox\_extend\_0 | bbox\_extend\_5 | bbox\_extend\_10 |
| 20000 | box | 0.8422 | 0.8582 | 0.8068 |
| mask | 0.8347 | 0.8406 | 0.8154 |
| 30000 | box | 0.8727 | 0.8787 | 0.8763 |
| mask | 0.8551 | 0.8470 | 0.8508 |
| 50000 | box | 0.9189 | 0.9054 | 0.9150 |
| mask | 0.8754 | 0.8750 | 0.8655 |
| 75999 | box | 0.9370 | 0.9339 | 0.9248 |
| mask | 0.8913 | 0.8866 | 0.8780 |
| 100000 | box | 0.9616 | 0.9421 | 0.9510 |
| mask | 0.8976 | 0.8893 | 0.8871 |

2．对测试集进行验证

模型数据bbox扩充—0，各个测试集（bbox扩充0，5，10）对其的测试结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 迭代次数 | Task | Minival  bbox\_extend\_0 | Minival  bbox\_extend\_5 | Minival  bbox\_extend\_10 |
| 20000 | box | 0.5513 | 0.4771 | 0.3873 |
| mask | 0.5801 | 0.5801 | 0.5801 |
| 30000 | box | 0.5897 | 0.5398 | 0.4364 |
| mask | 0.6147 | 0.6147 | 0.6147 |
| 50000 | box | 0.5508 | 0.4909 | 0.3930 |
| mask | 0.5920 | 0.5920 | 0.5920 |
| 75999 | box | 0.5252 | 0.4549 | 0.3668 |
| mask | 0.5632 | 0.5632 | 0.5632 |
| 100000 | box | 0.5263 | 0.4568 | 0.3683 |
| mask | 0.5674 | 0.5674 | 0.5674 |

模型数据bbox扩充—0，各个测试集（bbox扩充0，5，10）对其的测试结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 迭代次数 | Task | Minival  bbox\_extend\_0 | Minival  bbox\_extend\_5 | Minival  bbox\_extend\_10 |
| 20000 | box | 0.5073 | 0.5374 | 0.4760 |
| mask | 0.5868 | 0.5868 | 0.5868 |
| 30000 | box | 0.5396 | 0.5281 | 0.4502 |
| mask | 0.6103 | 0.6103 | 0.6103 |
| 50000 | box | 0.5021 | 0.5129 | 0.4408 |
| mask | 0.5735 | 0.5735 | 0.5735 |
| 75999 | box | 0.5208 | 0.5495 | 0.4747 |
| mask | 0.5878 | 0.5878 | 0.5878 |
| 100000 | box | 0.5352 | 0.5469 | 0.4605 |
| mask | 0.6099 | 0.6099 | 0.6099 |

模型数据bbox扩充—0，各个测试集（bbox扩充0，5，10）对其的测试结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 迭代次数 | Task | Minival  bbox\_extend\_0 | Minival  bbox\_extend\_5 | Minival  bbox\_extend\_10 |
| 20000 | box | 0.4072 | 0.5019 | 0.5248 |
| mask | 0.5705 | 0.5705 | 0.5705 |
| 30000 | box | 0.4116 | 0.4972 | 0.5333 |
| mask | 0.5771 | 0.5771 | 0.5771 |
| 50000 | box | 0.4121 | 0.5009 | 0.5014 |
| mask | 0.5512 | 0.5512 | 0.5512 |
| 75999 | box | 0.4273 | 0.5032 | 0.5089 |
| mask | 0.5810 | 0.5810 | 0.5810 |
| 100000 | box | 0.4433 | 0.5322 | 0.5394 |
| mask | 0.5821 | 0.5821 | 0.5821 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 迭代次数 | Task | bbox\_extend\_0 | bbox\_extend\_5 | bbox\_extend\_10 |
| 20000 | box | 0.5513 | 0.5374 | 0.5248 |
| mask | 0.5801 | 0.5868 | 0.5705 |
| 30000 | box | 0.5897 | 0.5281 | 0.5333 |
| mask | 0.6147 | 0.6103 | 0.5771 |
| 50000 | box | 0.5508 | 0.5129 | 0.5014 |
| mask | 0.5920 | 0.5735 | 0.5512 |
| 75999 | box | 0.5252 | 0.5495 | 0.5089 |
| mask | 0.5632 | 0.5878 | 0.5810 |
| 100000 | box | 0.5263 | 0.5469 | 0.5394 |
| mask | 0.5674 | 0.6099 | 0.5821 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 迭代次数 | Task | bbox\_extend\_0 | | bbox\_extend\_5 | | bbox\_extend\_10 | |
| 验证集 | 测试集 | 验证集 | 测试集 | 验证集 | 测试集 |
| 20000 | box | 0.8422 | 0.5513 | 0.8582 | 0.5374 | 0.8068 | 0.5248 |
| mask | 0.8347 | 0.5801 | 0.8406 | 0.5868 | 0.8154 | 0.5705 |
| 30000 | box | 0.8727 | 0.5897 | 0.8787 | 0.5281 | 0.8763 | 0.5333 |
| mask | 0.8551 | 0.6147 | 0.8470 | 0.6103 | 0.8508 | 0.5771 |
| 50000 | box | 0.9189 | 0.5508 | 0.9054 | 0.5129 | 0.9150 | 0.5014 |
| mask | 0.8754 | 0.5920 | 0.8750 | 0.5735 | 0.8655 | 0.5512 |
| 75999 | box | 0.9370 | 0.5252 | 0.9339 | 0.5495 | 0.9248 | 0.5089 |
| mask | 0.8913 | 0.5632 | 0.8866 | 0.5878 | 0.8780 | 0.5810 |
| 100000 | box | 0.9616 | 0.5263 | 0.9421 | 0.5469 | 0.9510 | 0.5394 |
| mask | 0.8976 | 0.5674 | 0.8893 | 0.6099 | 0.8871 | 0.5821 |

**fc: 81\*14\*14**

**conv: 256\*14\*14**

mask：

位置：**512\*324（324=81\*4）**

类别：**512\*81（81类）**