**RFCN使用说明**

# 环境

**代码地址**：<https://github.com/YuwenXiong/py-R-FCN>

**容器：** caffe\_gpu\_rfcn\_ys (213机)

nvidia-docker run -idt -v /data:/opt --name caffe\_gpu\_ rfcn\_ys caffe\_ssd

**代码路径：**/data/yushan/py-R-FCN

作者明确指明PSROIPooling层不支持CPU，因此这次实验在GPU中进行。

## 1. 编译

1）cd $RFCN\_ROOT/lib

Make

2）Build Caffe and pycaffe

cd $RFCN\_ROOT/caffe

make -j8 && make pycaffe

直接用GPU编译：

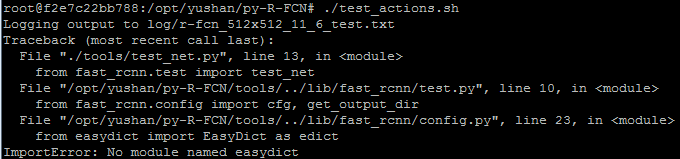
WITH\_PYTHON\_LAYER := 1

USE\_CUDNN := 1

## 2. 存在问题

**执行：./test\_actions.sh**

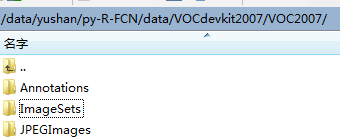
**错误如下：**



**解决办法：**pip install easydict

# 二、准备VOC数据

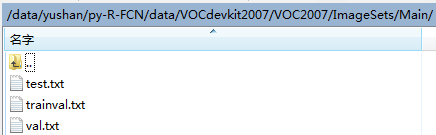
## 1. 数据存放格式



**Annotations：存放xml**

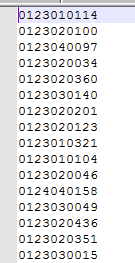
**JPEGImages：存放图片**

**ImageSets：存放txt，如下图所示：**



**备注：**

txt中存放图片的名字（图片名与xml名是一样的），如下图所示：



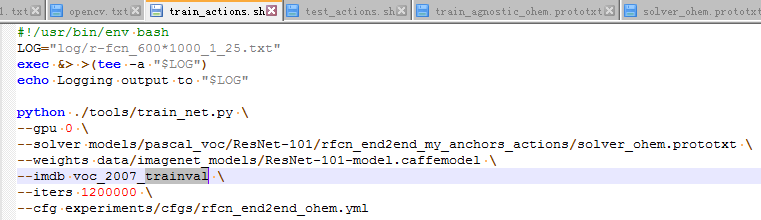
## 2. 生成txt

**存放位置：**/data/yushan/caffe/data/actions/ index\_for\_RFCN.py

**备注：**代码中将（test.txt、trainval.txt、train.txt、val.txt）这四个名字写死了，所以生成的txt名字需保证与这四个名字一样，可以少但是不能不一样。

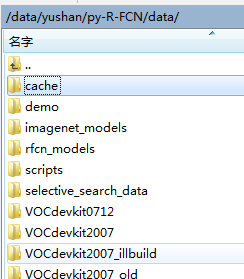
# 三、训练

## 1. 训练脚本：train\_actions.sh



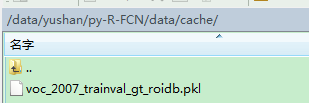
### 参数说明：

1. 如果训练数据是trainval.txt，参数--imdb 对应的是voc\_2007\_trainval；
2. 只支持参数—weights，保存模型的索引会从头开始；不支持参数—snapshot。
3. 如果训练数据改变，需要将下面图中的cache文件夹删除。



**解释：**

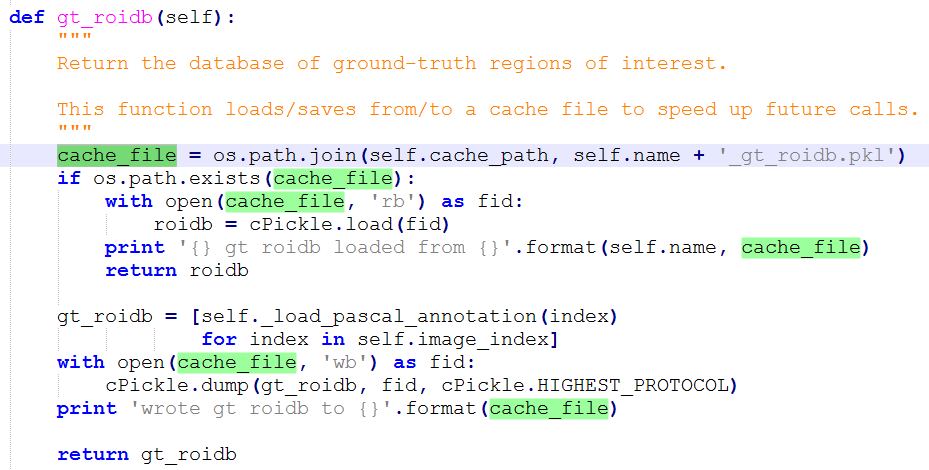
cache中保存了原始的标注信息，是GT的缓存。具体如下：



原因是data/cache下存在\*\_gt\_roidb.pkl文件，就会直接从这个文件中读取信息，并没有读取新图片和新标注信息。

pascal\_voc.py的gt\_roidb函数中有这部分的流程处理。如果data/cache下存在\*\_gt\_roidb.pkl文件，就会直接从这个文件中读取信息；如果data/cache下不存在\*\_gt\_roidb.pkl文件，才会按照索引文件中的文件名依次使用\_load\_pascal\_annotation读取相关的roi信息。

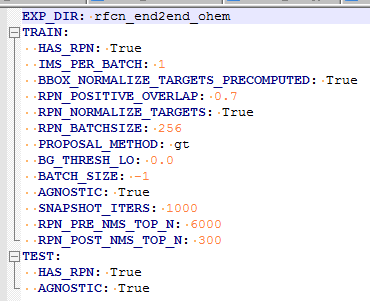
pascal\_voc.py的gt\_roidb函数中这部分的处理代码如下：



1. 参数—cfg不需要修改，写成这样就行；

**位置：**/data/yushan/py-R-FCN/**experiments/cfgs/ rfcn\_end2end\_ohem.yml**

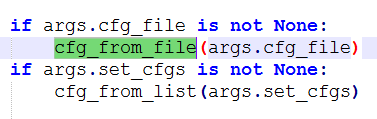
**内容如下：**



**解释：**

必须要提供—cfg对应的yml文件，否则会提示需要提供selective search方法的.mat文件。rfcn\_end2end\_ohem.yml里面也包含了HAS\_RPN等参数的设置。

虽然config.py中有\_\_C.TEST.HAS\_RPN = False，\_\_C.TRAIN.HAS\_RPN = False，但是实际上最终是用yml中配置的HAS\_RPN。因为在test\_net.py中的main函数中有如下的设置：



上面的args.cfg\_file就是—cfg指定的yml文件，如果提供了cfg对应的yml文件， 那么cfg\_from\_file函数会从yml文件中读取配置，从而更改config.py中相关变量的值。

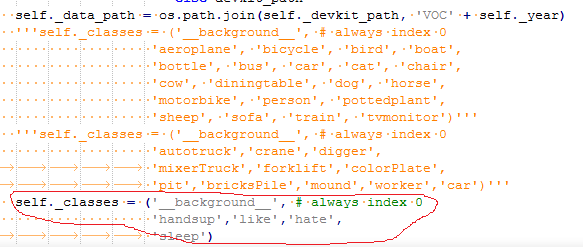
也就是说yml文件是对外的配置文件，可以通过改yml直接修改配置。config.py和yml文件中同名的变量，最后会以yml文件中的配置为准。

## 2. 修改类别

**5类**：'\_\_background\_\_', 'handsup', 'like', 'hate', 'sleep'

### 修改1：类别名称

**文件：**/data/yushan/py-R-FCN/lib/datasets/**pascal\_voc.py**



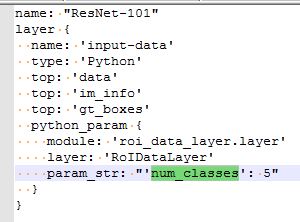
### 修改2：类别个数

**文件：**

/data/yushan/py-R-FCN/models/pascal\_voc/ResNet101/

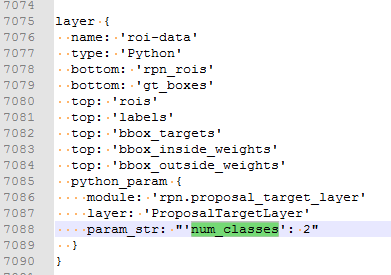
rfcn\_end2end\_my\_anchors\_actions/ **train\_agnostic\_ohem.prototxt**

**数据输入层 RoIDataLayer：**



**备注：**

下图中的“num\_classes”参数为2（前景、背景），不需要修改。



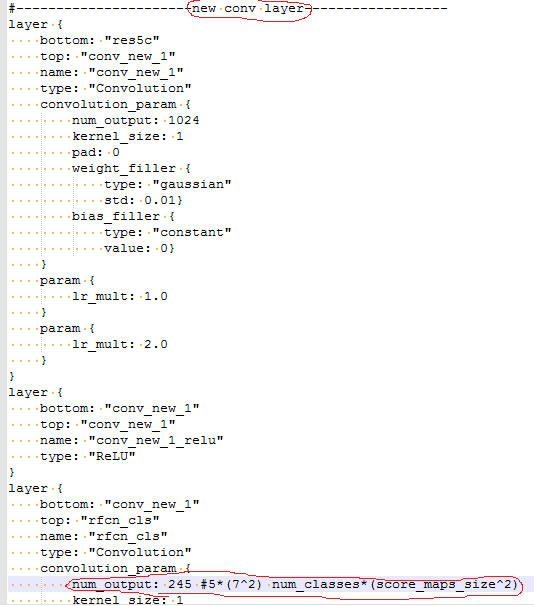
### 修改3：与类别个数有关的

**文件：**

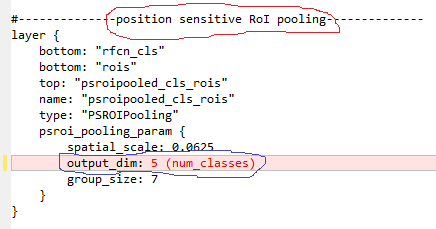
/data/yushan/py-R-FCN/models/pascal\_voc/ResNet101/

rfcn\_end2end\_my\_anchors\_actions/ **train\_agnostic\_ohem.prototxt**

#### 修改1）：检测的类别分类层，卷积核个数=类别数\*maps\_size\*maps\_size；



#### 修改2）：检测的类别分类后的PSROIPooling层，output\_dim为类别数；



## 3. 修改anchor

修改anchors，**除了修改原来的网络结构外，还需要修改代码**。因为原来的代码很多地方写死了，并没有暴露在外面的接口去设置。

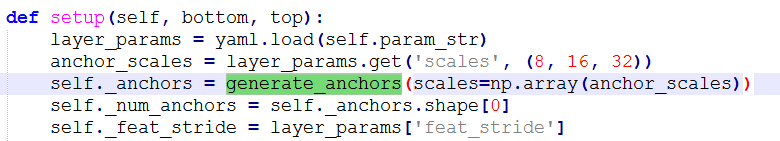
### 3.1代码部分修改

#### 1）修改AnchorTargetLayer和ProposalLayer 层的代码：

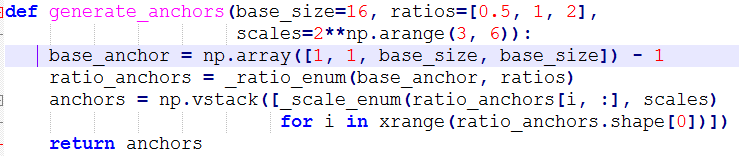
**文件位置**：/data/yushan/py-R-FCN/lib/**rpn/ anchor\_target\_layer.py**

/data/yushan/py-R-FCN/lib/**rpn/proposal\_layer.py**

AnchorTargetLayer和ProposalLayer 类的setup函数的前半部分的代码是一样的，如下图：



这里的setup函数没有给generate\_anchors 函数传入ratios参数。generate\_anchors 函数默认的ratios参数为[0.5,1,2]。如下图：



因此，如果要修改anchors的scales和ratio，需要修改AnchorTargetLayer和ProposalLayer 类的setup函数的代码。

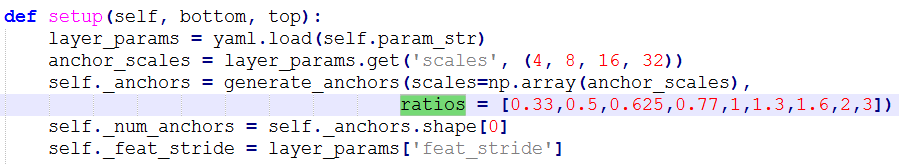
**例如这里使用4个scales，9个ratios。**

**需要修改以下两项：**

**Scales：**增加一个维度，变为(4,8,16,32)。即对应[16\*4,16\*8,16\*16,16\*32]，16是默认的base\_size。

**Ratios：**需要给generate\_anchors 函数传入ratios参数。传入的ratios参数为[0.33,0.5,0.625,0.77,1,1.3,1.6,2,3]。

修改后的代码如下图：



**修改之后的anchors个数就从之前的9个（3\*3）变成了36个（4\*9）。**

**备注：**proposal\_layer.py、anchor\_target\_layer.py的setup函数都要改，不然会错误。

### 3.2修改train\_agnostic\_ohem.prototxt

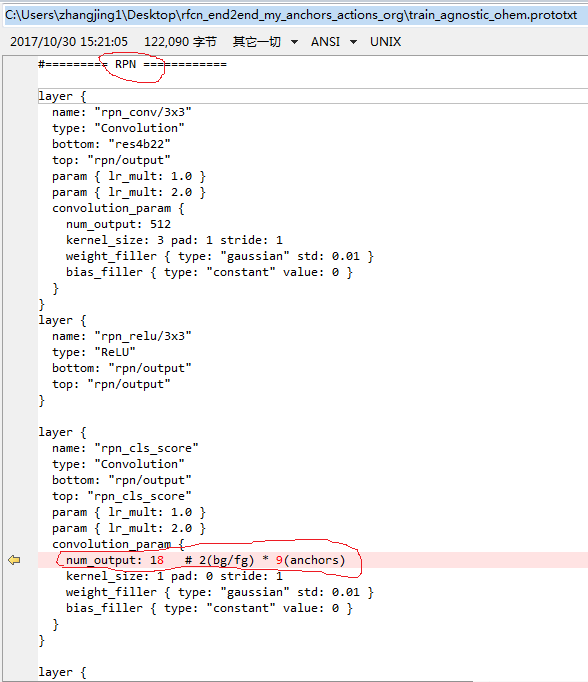
**文件位置：**

/data/yushan/py-R-FCN/models/pascal\_voc/ResNet101/

rfcn\_end2end\_my\_anchors\_actions\_org/ train\_agnostic\_ohem.prototxt

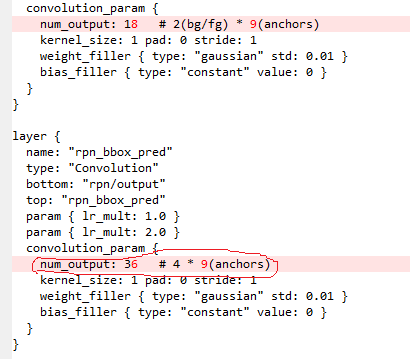
#### 修改1：rpn的类别分数层

**"rpn\_cls\_score"：num\_output: 18 # 2(bg/fg) \* 9(anchors)**



#### 修改2：rpn的位置回归层

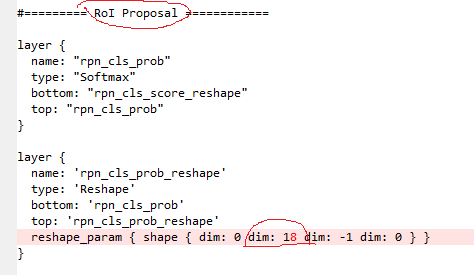
**"rpn\_bbox\_pred": num\_output: 36 # 4 \* 9(anchors)**



#### 修改3：rpn的类别分数softmax之后的reshape层

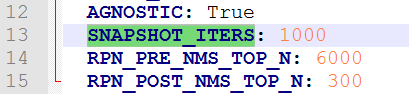
**'rpn\_cls\_prob\_reshape'：reshape\_param { shape { dim: 0 dim: 18 dim: -1 dim: 0 } }**

**备注**：18 = 2(bg/fg) \* 9(anchors)



## 4.设置模型保存的间隔

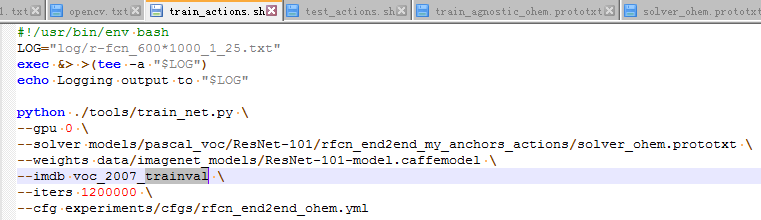
config.py中的\_\_C.TRAIN.SNAPSHOT\_ITERS 定义了模型snapshot的保存间。另外，在yml中也有设置了这个snapshot的保存间隔，如下图：（/data/yushan/py-R-FCN/experiments/cfgs/ rfcn\_end2end\_ohem**.yml** ）



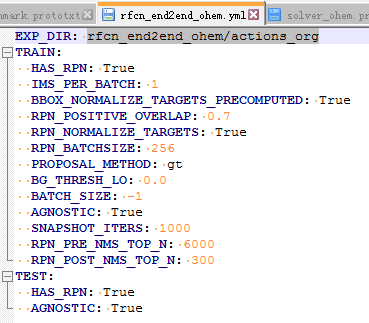
前面说过如果提供了cfg对应的yml文件，将会从yml文件中读取配置，从而更改config.py中的配置。最终以yml中配置的为准。

因此，这里只需要修改yml中SNAPSHOT\_ITERS，就可以控制模型的保存间隔了。

## 5. 设置模型保存的路径

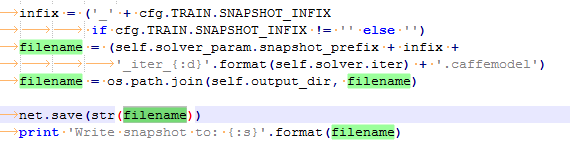


如果想将模型保存在” output/**rfcn\_end2end\_ohem/actions\_org**/voc\_2007\_trainval”文件夹下面，output是代码写死的，voc\_2007\_trainval是—imdb参数值，只需要修改” experiments/cfgs/rfcn\_end2end\_ohem.yml”文件中的参数**” EXP\_DIR”**就可以了，如下图所示：

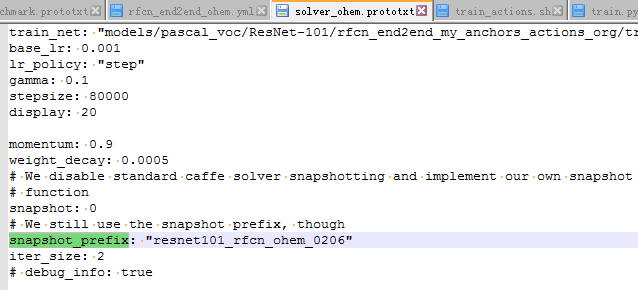


### 解释：

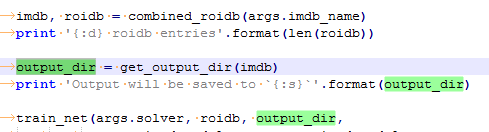
首先，**lib/fast\_rcnn/train.py**中filename是保存模型的路径，代码如下图所示：



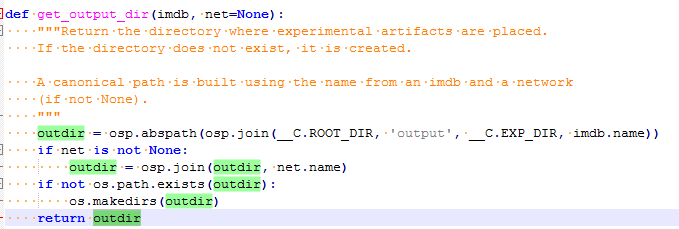
其中，self.solver\_param.snapshot\_prefix 是在**solver\_ohem.prototx**t文件中设置的，如下图所示：



**self.output\_dir** 是在**tools/train\_net.py**中获得的，如下图所示：



**get\_output\_dir()**函数是在**lib/fast\_rcnn/config.py**中，

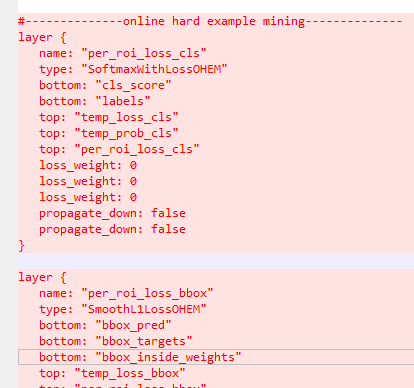


## 6.备注

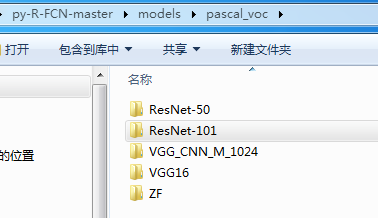
### 1. train\_agnostic\_ohem.prototxt与train\_agnostic.prototxt区别

train\_agnostic\_ohem.prototxt与train\_agnostic.prototxt基本一样，只是

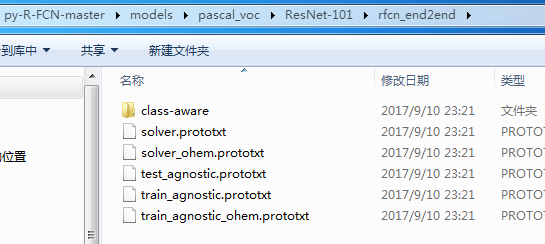
train\_agnostic\_ohem.prototxt多了一个困难样本处理，（rpn前景、背景样本均衡）如下图所示：



### 2. 支持的网络结构

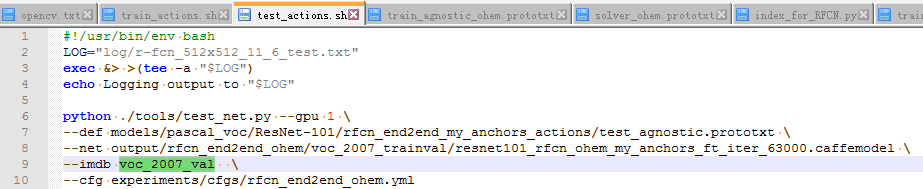


ResNet-101的网络结构文件如下图所示：（直接在github下载后就有这些网络结构文件）



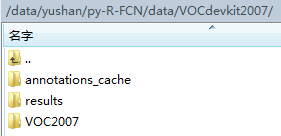
# 四、测试

## 1. 测试脚本：test\_actions.sh



### 参数说明：

1. 参数—cfg不需要修改，写成这样就行；
2. 如果测试数据是val.txt，参数--imdb 对应的是voc\_2007\_val；
3. 如果测试数据改变，需要将下图中annotations\_cache文件夹删除。



**解释：**annotations\_cache中保存了原始的标注信息，是GT的缓存。图片名为索引的关键字。因为如果annotations\_cache中已经保存了信息，当我们再次执行测试程序时，是不会重新保存的。

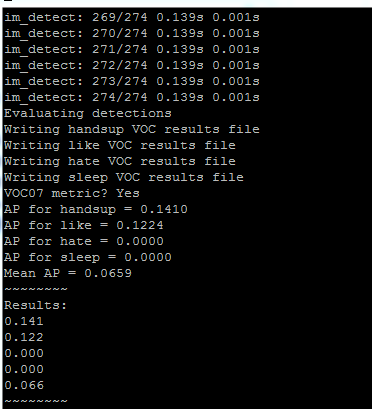
如果之前运行过测试，然后又在data\VOCdevkit2007\VOC2007新增了一批图片数据，此时是需要删除data/VOCdevkit2007/annotations\_cache目录的，否则会在图片测试结束后，计算mAP值时报错，找不到新增加的图片，从而提示关键字索引出错。

1. data\VOCdevkit2007下需要创建results/VOC2007/Main文件夹。

**解释：**测试时，会先依次测试每一张图片，然后按照类别将测试的结果保存到results/VOC2007/Main文件夹下面。测试时不会主动创建results/VOC2007/Main文件夹，如果不在VOCdevkit2007文件夹下面手动创建这些文件夹，就会在测试的结果保存时，提示结果文件创建失败。下图是保存成功的测试结果：

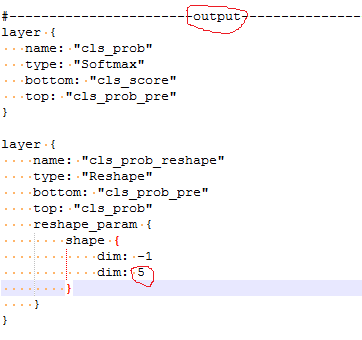


### 结果：（mAP值）



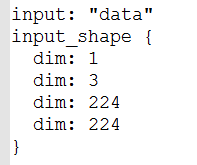
## 2. 修改类别

除了修改上面的[类别分类层（卷积核个数=类别数\*maps\_size\*maps\_size）](#_修改1）：检测的类别分类层，卷积核个数=类别数*maps_size*m)和[类别分类后的PSROIPooling层（output\_dim为类别数）](#_修改2）：检测的类别分类后的PSROIPooling层，output_)外，还需要修改类别分数对应的Reshape层。如下图：**(5类)**



## 3. 测试时的尺度问题

test\_agnostic.prototxt中一开始就是设置input\_shape的地方，如下图：



**但是，实际上这个设置并不能控制测试时的图片尺寸**。

R-FCN测试时的图片尺寸是由**cfg.TEST.SCALES或者cfg.TEST.MAX\_SIZE的设置唯一决定**的。

R-FCN中测试的总入口是**lib/ fast\_rcnn/test.py**文件中的im\_detect函数。

im\_detect函数的第一行就调用了\_get\_blobs函数，但是并没有设置Resize图片的target\_size。如下图：



最终Resize图片是在\_get\_image\_blob函数中。在\_get\_image\_blob函数中，会先根据图片的尺寸和cfg.TEST.SCALES或者cfg.TEST.MAX\_SIZE设置的尺寸计算图片的缩放比例，保证缩放后的图片会有一个尺寸为cfg.TEST.SCALES或者cfg.TEST.MAX\_SIZE设置的尺寸。这里的**缩放并不改变原图的宽高比例**，也就是说最终Resize后的图片和原图的宽高比例一样。

因此，最终图片的尺寸是由cfg.TEST.SCALES或者cfg.TEST.MAX\_SIZE设置的，并不是由test\_agnostic.prototxt中input\_shape设置的。

## 4. 在图上显示检测结果（还未完成）

# 五、论文

R-FCN: Object Detection via Region-based Fully Convolutional Networks

**论文：**<https://arxiv.org/abs/1605.06409> 2016年、[何恺明](http://kaiminghe.com/)

**论文解释：**<http://blog.csdn.net/baidu_32173921/article/details/71741970>