



(<http://lib.csdn.net/base/deeplearning>)

深度学习 (<http://lib.csdn.net/base/deeplearning>) - 深度学习应用 (<http://lib.csdn.net/deeplearning/node/747>) - 图像检测 (<http://lib.csdn.net/deeplearning/knowledge/1726>)

👁 1585 💬 22

Faster RCNN代码理解 (Python)

作者：u011956147 (<http://my.csdn.net/u011956147>)

最近开始学习深度学习，看了下Faster RCNN的代码，在学习的过程中也查阅了很多其他人写的博客，得到了很大的帮助，所以也打算把自己一些粗浅的理解记录下来，一是记录下自己的菜鸟学习之路，方便自己过后查阅，二来可以回馈网络。目前编程能力有限，且是第一次写博客，中间可能会有一些错误。

目录

目录

第一步准备

第二步Stage 1 RPN init from ImageNet model

在config参数的基础上改动参数以适合当前任务主要有

初始化化caffe

准备roidb和imdb

设置输出路径output_dir get_output_dirimdb函数在config中用来保存中间生成的caffemodule等

正式开始训练

保存最后得到的权重参数

第三步Stage 1 RPN generate proposals

关注rpn_generate函数

保存得到的proposal文件

第四步Stage 1 Fast R-CNN using RPN proposals init from ImageNet model

第五步Stage 2 RPN init from stage 1 Fast R-CNN model

第六步Stage 2 RPN generate proposals

第七步Stage 2 Fast R-CNN init from stage 2 RPN R-CNN model

第八步输出最后模型

AnchorTargetLayer和ProposalLayer

代码文件夹说明

tools

RPN

nms

参考

原文地址

第一步，准备

从train_faster_rcnn_alt_opt.py入：

初始化参数：args = parse_args() 采用的是Python的argparse

主要有-net_name, -gpu, -cfg等（在cfg中只是修改了几个参数，其他大部分参数在config.py中，涉及到训练整个网络）。

cfg_from_file(args.cfg_file) 这里便是代用config中的函数cfg_from_file来读取前面cfg文件中的参数，同时调用_merge_a_into_b函数把所有的参数整合，其中__C = edict() cfg = __C cfg是一个词典（edict）数据结构。
faster rcnn采用的是多进程，mp_queue是进程间用于通讯的数据结构

```
1 import multiprocessing as mp
2 mp_queue = mp.Queue()
```

同时solvers, max_iters, rpn_test_prototxt = get_solvers(args.net_name)得到solver参数

接下来便进入了训练的各个阶段。

第二步, Stage 1 RPN, init from ImageNet model

```
1  cfg.TRAIN.SNAPSHOT_INFIX = 'stage1'
2  mp_kwargs = dict(
3      queue=mp_queue,
4      imdb_name=args.imdb_name,
5      init_model=args.pretrained_model,
6      solver=solvers[0],
7      max_iters=max_iters[0],
8      cfg=cfg)
9  p = mp.Process(target=train_rpn, kwargs=mp_kwargs)
10 p.start()
11 rpn_stage1_out = mp_queue.get()
12 p.join()
```

可以看到第一个步骤是用ImageNet的模型M0来Finetuning RPN网络得到模型M1。以训练为例, 这里的args参数都在脚本 experiments/scripts/faster_rcnn_alt_opt.sh中找到。主要关注train_rpn函数。

对于train_rpn函数, 主要分一下几步:

1. 在config参数的基础上改动参数, 以适合当前任务, 主要有

```
1  cfg.TRAIN.HAS_RPN = True
2  cfg.TRAIN.BBOX_REG = False # applies only to Fast R-CNN bbox regression
3  cfg.TRAIN.PROPOSAL_METHOD = 'gt'
```

这里, 关注proposal method 使用的是gt, 后面会使用到gt_roidb函数, 重要。

2. 初始化caffe

3. 准备roidb和imdb

主要涉及到的函数get_roidb

在get_roidb函数中调用factory中的get_imdb根据__sets[name]中的key (一个lambda表达式) 转到pascal_voc类。class pascal_voc (imdb) 在初始化自己的时候, 先调用父类的初始化方法, 例如:

```

1  {
2      year:'2007'
3      image _set:'trainval'
4      devkit _path:'data/VOCdevkit2007'
5      data _path:'data /VOCdevkit2007/VOC2007'
6      classes:(...)_如果想要训练自己的数据, 需要修改这里_
7      class _to _ind:{...} _一个将类名转换成下标的字典 _ 建立索引0,1,2....
8      image _ext:'.jpg'
9      image _index: ['000001', '000003', .....]_根据trainval.txt获取到的image索引_
10     roidb _handler: <Method gt_roidb >
11     salt: <Object uuid >
12     comp _id:'comp4'
13     config:{...}
14 }
```

注意, 在这里, 并没有读入任何数据, 只是建立了图片的索引。

```
1  imdb.set_proposal_method(cfg.TRAIN.PROPOSAL_METHOD)
```

设置proposal方法, 接上面, 设置为gt, 这里只是设置了生成的方法, 第一次调用发生在下一句, roidb =

get_training_roidb(imdb) -> append_flipped_images()时的这行代码: “boxes = self.roidb[i]['boxes'].copy()”, 其中

get_training_roidb位于train.py, 主要实现图片的水平翻转, 并添加回去。实际是该函数调用了imdb.

append_flipped_images也就是在这个函数, 调用了pascal_voc中的gt_roidb, 转而调用了同一个文件中的

_load_pascal_annotation, 该函数根据图片的索引, 到Annotations这个文件夹下去找相应的xml标注数据, 然后加载

所有的bounding box对象, xml的解析到此结束, 接下来是roidb中的几个类成员的赋值:

boxes 一个二维数组, 每一行存储 xmin ymin xmax ymax

gt_classes存储了每个box所对应的类索引(类数组在初始化函数中声明)

gt_overlap是一个二维数组，共有num_classes(即类的个数)行，每一行对应的box的类索引处值为1，其余皆为0，后来被转成了稀疏矩阵

seg_areas存储着某个box的面积

flipped 为false 代表该图片还未被翻转(后来在train.py里会将翻转的图片加进去，用该变量用于区分

最后将这些成员变量组装成roidb返回。

在get_training_roidb函数中还调用了roidb中的prepare_roidb函数，这个函数就是用来准备imdb的roidb，给roidb中的字典添加一些属性，比如image(图像的索引)，width，height，通过前面的gt_overlap属性，得到max_classes和max_overlaps。

至此，

```
1 return roidb, imdb
```

4. 设置输出路径，output_dir = get_output_dir(imdb)，函数在config中，用来保存中间生成的caffemodule等

5.正式开始训练

```
1 model_paths = train_net(solver, roidb, output_dir,  
2                         pretrained_model=init_model,  
3                         max_iters=max_iters)
```

调用train中的train_net函数，其中，首先filter_roidb，判断roidb中的每个entry是否合理，合理定义为至少有一个前景box或背景box，roidb全是groudtruth时，因为box与对应的类的重合度(overlaps)显然为1，也就是说roidb起码要有一个标记类。如果roidb包含了一些proposal，overlaps在[BG_THRESH_LO, BG_THRESH_HI]之间的都将被认为是背景，大于FG_THRESH才被认为是前景，roidb至少要有有一个前景或背景，否则将被过滤掉。将没用的roidb过滤掉以后，返回的就是filtered_roidb。在train文件中，需要关注的是SolverWrapper类。详细见train.py，在这个类里面，引入了caffe SGDSlover，最后一句self.solver.net.layers[0].set_roidb(roidb)将roidb设置进layer(0)(在这里就是ROILayer)调用ayer.py中的set_roidb方法，为layer(0)设置roidb，同时打乱顺序。最后train_model。在这里，就需要去实例化每个层，在这个阶段，首先就会实现ROILayer，详细参考layer中的setup，在训练时roilayer的forward函

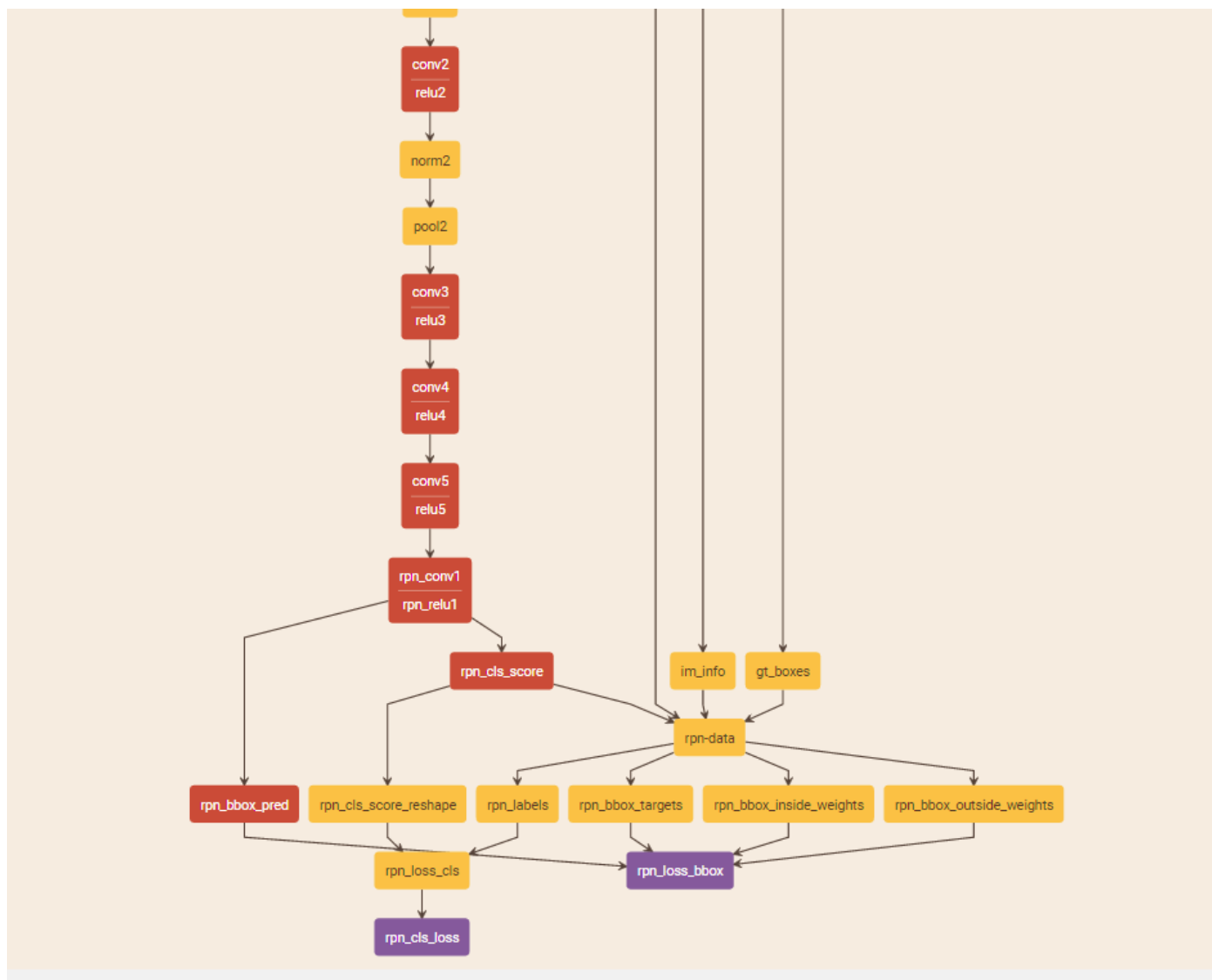
数，在第一个层，只需要进行数据拷贝，在不同的阶段根据prototxt文件定义的网络结构拷贝数据，blobs = self._get_next_minibatch()这个函数读取图片数据（调用get_minibatch函数，这个函数在minibatch中，主要作用是 faster rcnn 做实际的数据准备，在读取数据的时候，分出了boxes, gt_boxes, im_info（宽高缩放）等）。

第一个层，对于stage1_rpn_train.pt文件中，该layer只有3个top blob：'data'、'im_info'、'gt_boxes'。

对于stage1_fast_rcnn_train.pt文件中，该layer有6个top blob：top:

'data'、'rois'、'labels'、'bbox_targets'、'bbox_inside_weights'、'bbox_outside_weights'，这些数据准备都在 minibatch 中。至此数据便在caffe中流动了，直到训练结束。

画出网络的结构 (<http://ethereon.github.io/netscope/#/editor>) 这里只截取了一部分：



值得注意的是在rpn-data层使用的是AnchorTargetLayer，该层使用python实现的，往后再介绍。

6.保存最后得到的权重参数

```
1 rpn_stage1_out = mp_queue.get()
```

至此，第一阶段完成，在后面的任务开始时，如果有需要，会在这个输出的地址找这一阶段得到的权重文件。

第三步，Stage 1 RPN, generate proposals

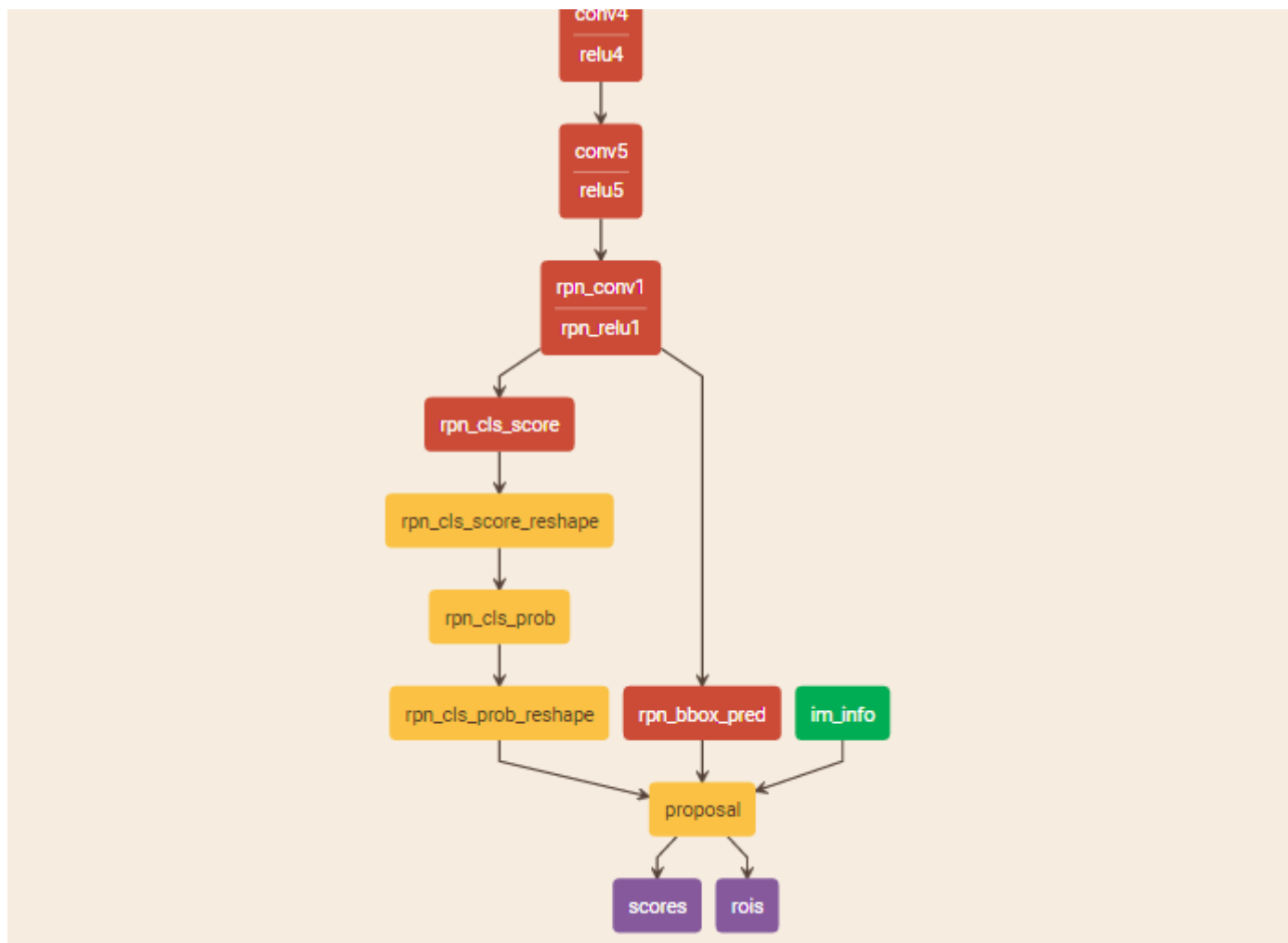
这一步就是调用上一步训练得到的模型M1来生成proposal P1，在这一步只产生proposal，参数：

```
1 mp_kwargs = dict(  
2     queue=mp_queue,  
3     imdb_name=args.imdb_name,  
4     rpn_model_path=str(rpn_stage1_out['model_path']),  
5     cfg=cfg,  
6     rpn_test_prototxt=rpn_test_prototxt)  
7 p = mp.Process(target=rpn_generate, kwargs=mp_kwargs)  
8 p.start()  
9 rpn_stage1_out['proposal_path'] = mp_queue.get()['proposal_path']  
10 p.join()
```

1.关注rpn_generate函数

前面和上面讲到的train_rpn基本相同，从rpn_proposals = imdb_proposals(rpn_net, imdb)开始，imdb_proposals函数在rpn.generate.py文件中,rpn_proposals是一个列表的列表，每个子列表。对于imdb_proposals，使用im = cv2.imread(imdb.image_path_at(i))读入图片数据，调用 im_proposals生成单张图片的rpn proposals，以及得分。这里，im_proposals函数会调用网络的forward，从而得到想要的boxes和scores，这里需要好好理解blobs_out =

net.forward(data,im_info)中net forward和layer forward间的调用关系。



在这里，也会有proposal，同样会使用python实现的ProposalLayer，这个函数也在rpn文件夹内，后面再补充。

```
1 boxes = blobs_out['rois'][:, 1:].copy() / scale
2 scores = blobs_out['scores'].copy()
3 return boxes, scores
```

至此，得到imdb proposal

2.保存得到的proposal文件

```
1 queue.put({'proposal_path': rpn_proposals_path})
2 rpn_stage1_out['proposal_path'] = mp_queue.get()['proposal_path']
```

至此，Stage 1 RPN, generate proposals结束

第四步，Stage 1 Fast R-CNN using RPN proposals, init from ImageNet model

参数：

```
1 cfg.TRAIN.SNAPSHOT_INFIX = 'stage1'
2 mp_kwargs = dict(
3     queue=mp_queue,
4     imdb_name=args.imdb_name,
5     init_model=args.pretrained_model,
6     solver=solvers[1],
7     max_iters=max_iters[1],
8     cfg=cfg,
9     rpn_file=rpn_stage1_out['proposal_path'])
10 p = mp.Process(target=train_fast_rcnn, kwargs=mp_kwargs)
11 p.start()
12 fast_rcnn_stage1_out = mp_queue.get()
13 p.join()
```

这一步，用上一步生成的proposal，以及imagenet模型M0来训练fast-rcnn模型M2。

关注train_fast_rcnn

同样地，会设置参数，这里注意cfg.TRAIN.PROPOSAL_METHOD = 'rpn' 不同于前面，后面调用的将是rpn_roidb。

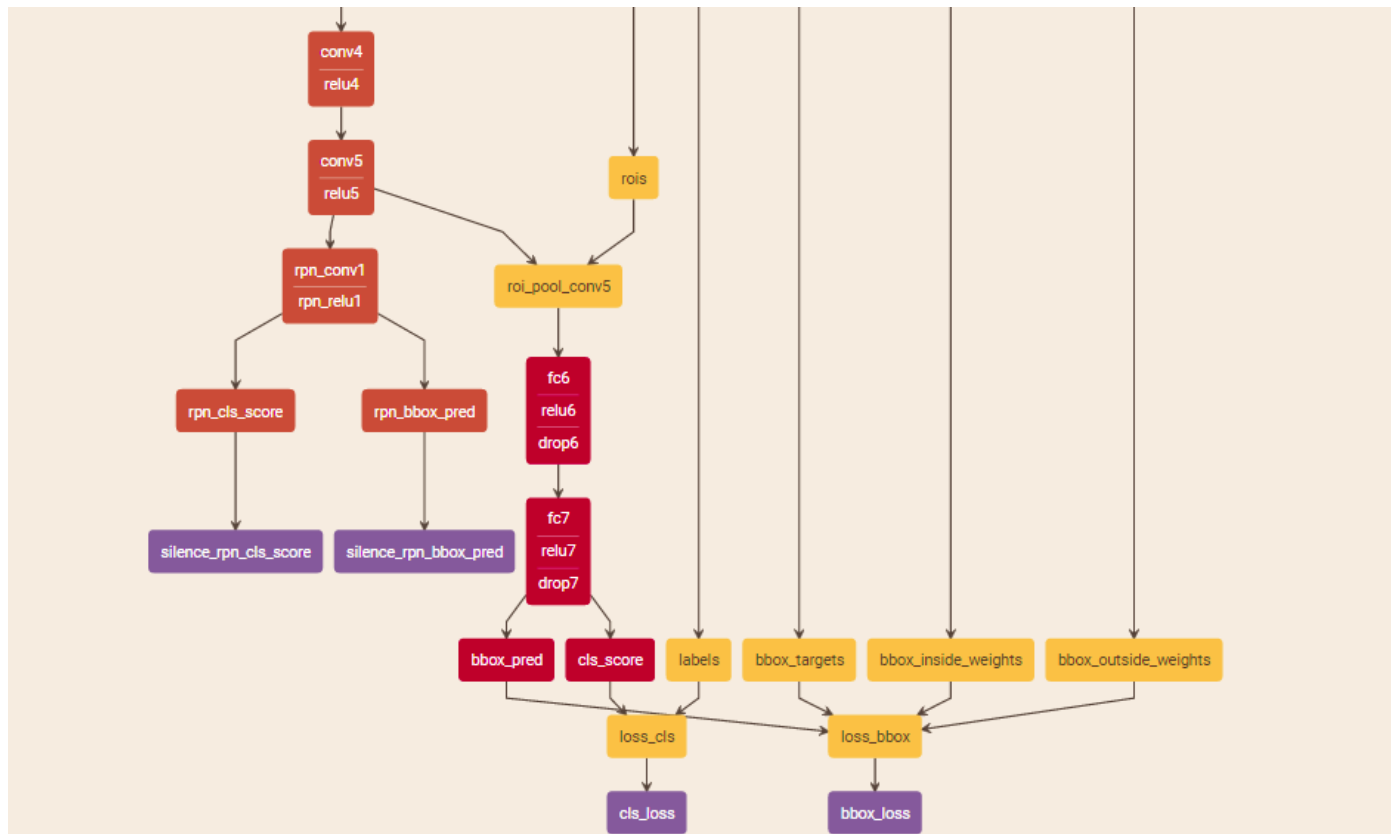
cfg.TRAIN.IMS_PER_BATCH = 2，每个mini-batch包含两张图片，以及它们proposal的roi区域。且在这一步是有

rpn_file的（后面和rpn_roidb函数使用有关）。其他的和前面差不多。提一下，这里在train_net的时候，会调用

`add_bbox_regression_targets`位于`roidb`中，主要是添加bbox回归目标，即添加`roidb`的‘`bbox_targets`’属性，同时根据`cfg`中的参数设定，求取`bbox_targets`的`mean`和`std`，因为需要训练`class-specific regressors`在这里就会涉及到`bbox_overlaps`函数，放在`util.bbox`中。

要注意的是在这一步`get_roidb`时，如前所说，使用的是`rpn_roidb`，会调用`imdb.create_roidb_from_box_list`该方法功能是从`box_list`中读取每张图的`boxes`，而这个`box_list`就是从上一步保存的`proposal`文件中读取出来的，然后做一定的处理，详细见代码，重点是在最后会返回`roidb`，`rpn_roidb`中的`gt_overlaps`是`rpn_file`中的`box`与`gt_roidb`中`box`的`gt_overlaps`等计算IoU等处理后得到的，而不像`gt_roidb()`方法生成的`gt_roidb`中的`gt_overlaps`全部为1.0。同时使用了

`imdb.merge_roidb`，类`imdb`的静态方法【这里不太懂，需要再学习下】，把`rpn_roidb`和`gt_roidb`归并为一个`roidb`，在这里，需要具体去了解合并的基本原理。

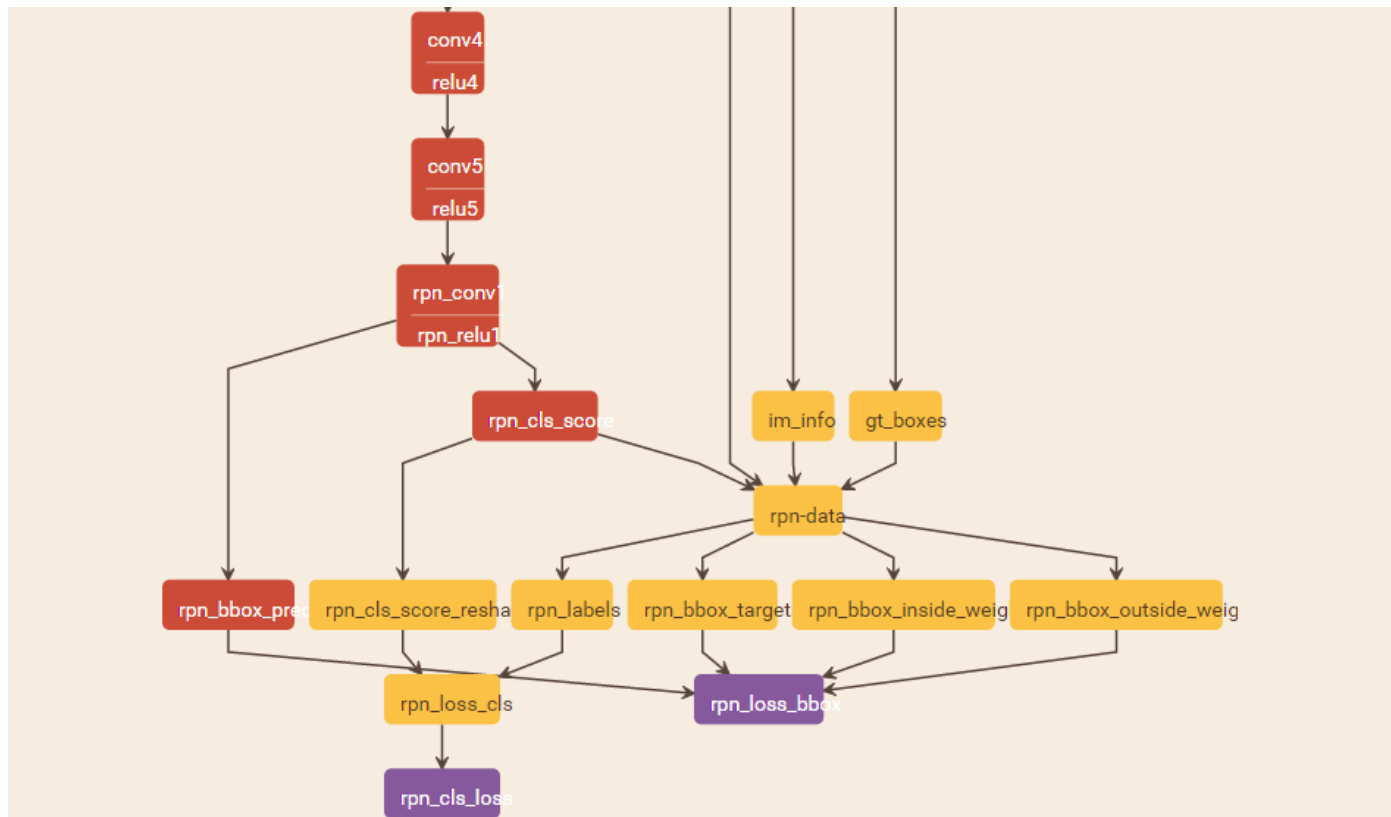


第五步，Stage 2 RPN, init from stage 1 Fast R-CNN model

参数：

```
1  cfg.TRAIN.SNAPSHOT_INFIX = 'stage2'
2  mp_kwargs = dict(
3      queue=mp_queue,
4      imdb_name=args.imdb_name,
5      init_model=str(fast_rcnn_stage1_out['model_path']),
6      solver=solvers[2],
7      max_iters=max_iters[2],
8      cfg=cfg)
9  p = mp.Process(target=train_rpn, kwargs=mp_kwargs)
10 rpn_stage2_out = mp_queue.get()
```

这部分就是利用模型M2练rpn网络，这一次与stage1的rpn网络不通，这一次conv层的参数都是不动的，只做前向计算，训练得到模型M3，这属于微调了rpn网络。

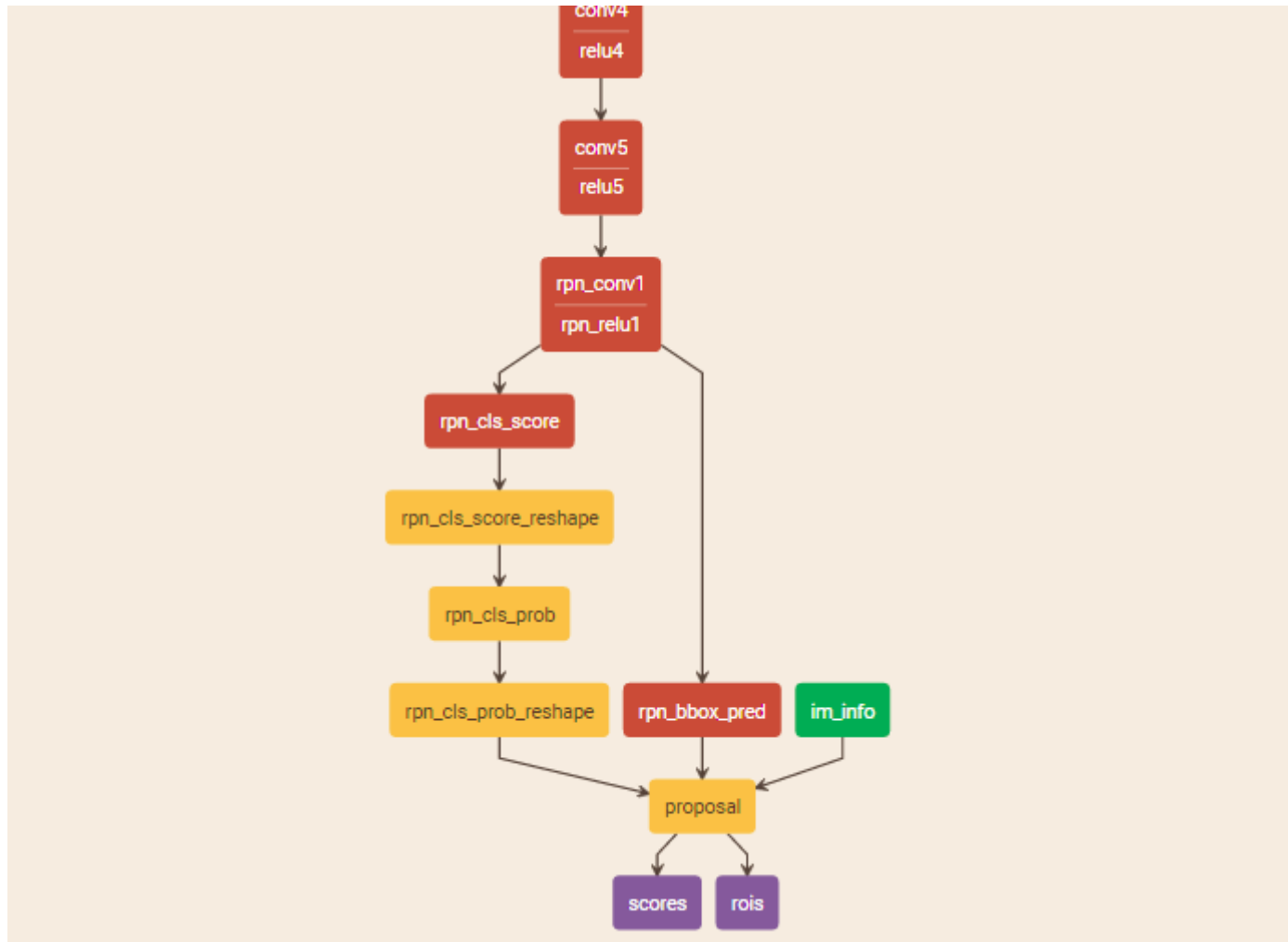


第六步，Stage 2 RPN, generate proposals

参数：

```
1 mp_kwargs = dict(  
2     queue=mp_queue,  
3     imdb_name=args.imdb_name,  
4     rpn_model_path=str(rpn_stage2_out['model_path']),  
5     cfg=cfg,  
6     rpn_test_prototxt=rpn_test_prototxt)  
7 p = mp.Process(target=rpn_generate, kwargs=mp_kwargs)  
8 p.start()  
9 rpn_stage2_out['proposal_path'] = mp_queue.get()['proposal_path']  
10 p.join()
```

这一步，基于上一步得到的M3模型，产生proposal P2，网络结构和前面产生proposal P1的一样。

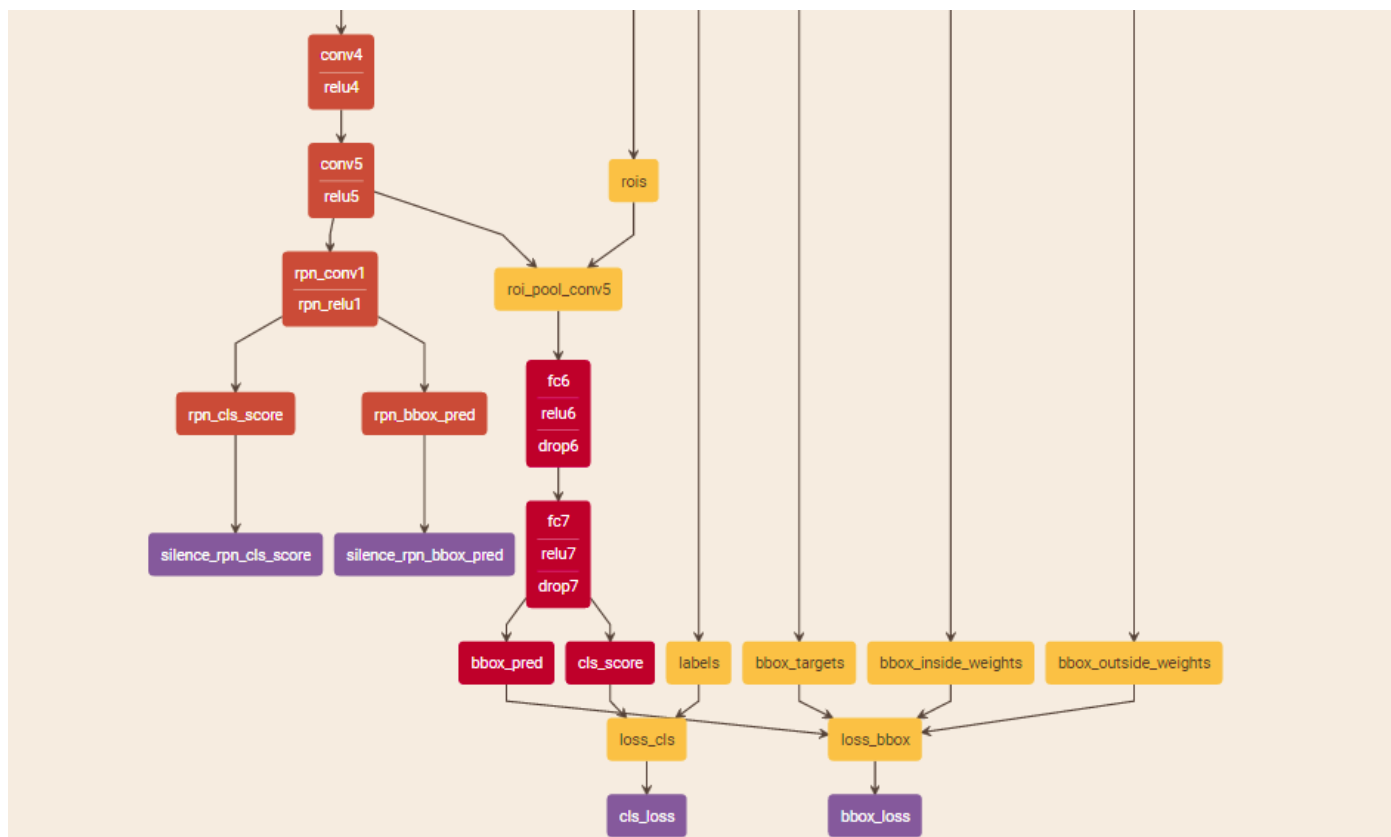


第七步，Stage 2 Fast R-CNN, init from stage 2 RPN R-CNN model

参数：


```
1  cfg.TRAIN.SNAPSHOT_INFIX = 'stage2'
2  mp_kwargs = dict(
3      queue=mp_queue,
4      imdb_name=args.imdb_name,
5      init_model=str(rpn_stage2_out['model_path']),
6      solver=solvers[3],
7      max_iters=max_iters[3],
8      cfg=cfg,
9      rpn_file=rpn_stage2_out['proposal_path'])
10 p = mp.Process(target=train_fast_rcnn, kwargs=mp_kwargs)
11 p.start()
12 fast_rcnn_stage2_out = mp_queue.get()
13 p.join()
```

这一步基于模型M3和P2训练fast rcnn得到最终模型M4，这一步，conv层和rpn都是参数固定，只是训练了rcnn层（也就是全连接层），与stage1不同，stage1只是固定了rpn层，其他层还是有训练。模型结构与stage1相同：



第八步，输出最后模型

```
1 final_path = os.path.join(
2     os.path.dirname(fast_rcnn_stage2_out['model_path']),
3     args.net_name + '_faster_rcnn_final.caffemodel')
4 print 'cp {} -> {}'.format(
5     fast_rcnn_stage2_out['model_path'], final_path)
6 shutil.copy(fast_rcnn_stage2_out['model_path'], final_path)
7 print 'Final model: {}'.format(final_path)
```

只是对上一步模型输出的一个拷贝。

至此，整个faster-rcnn的训练过程就结束了。

AnchorTargetLayer和ProposalLayer

前面说过还有这两个层没有说明，一个是anchortarget layer一个是proposal layer，下面逐一简要分析。

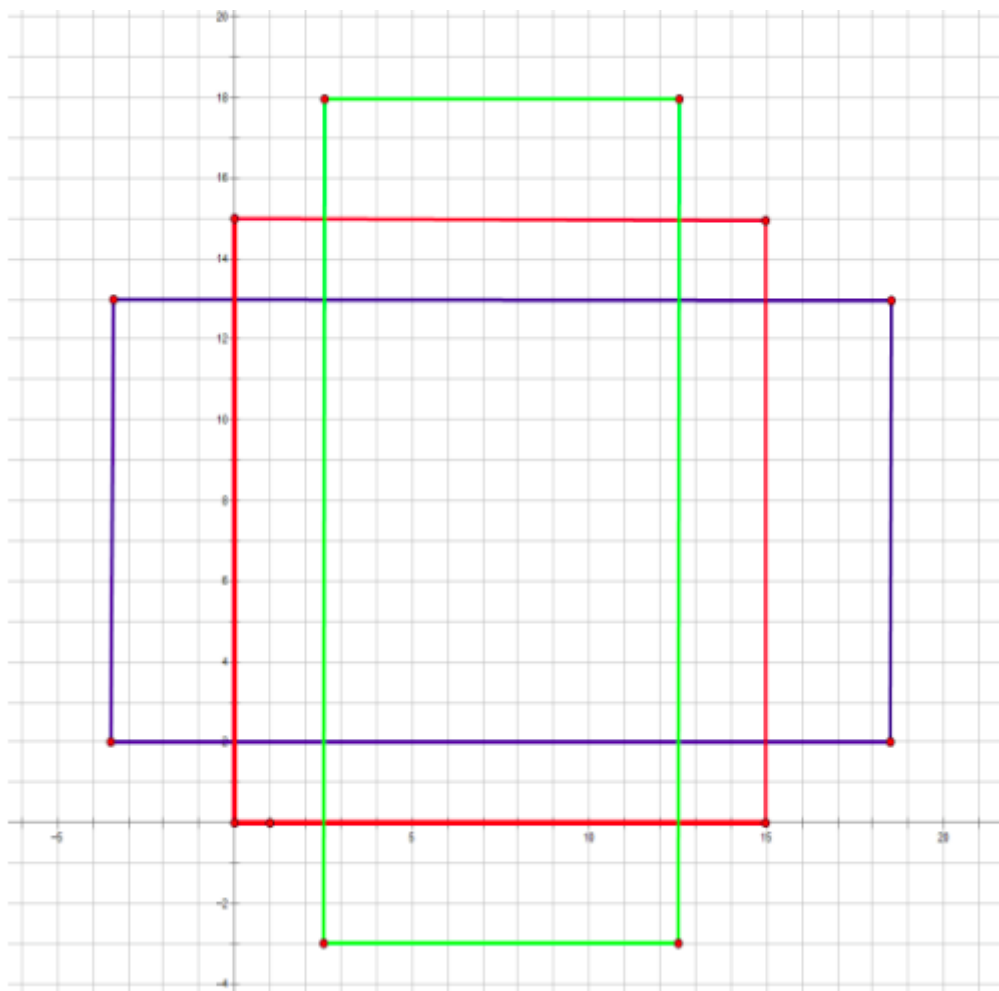
```
1 class AnchorTargetLayer(caffe.Layer)
```

首先是读取参数，在prototxt，实际上只读取了param_str: “‘feat_stride’: 16”，这是个很重要的参数，目前我的理解是滑块滑动的大小，对于识别物体的大小很有用，比如小物体的识别，需要把这个参数减小等。

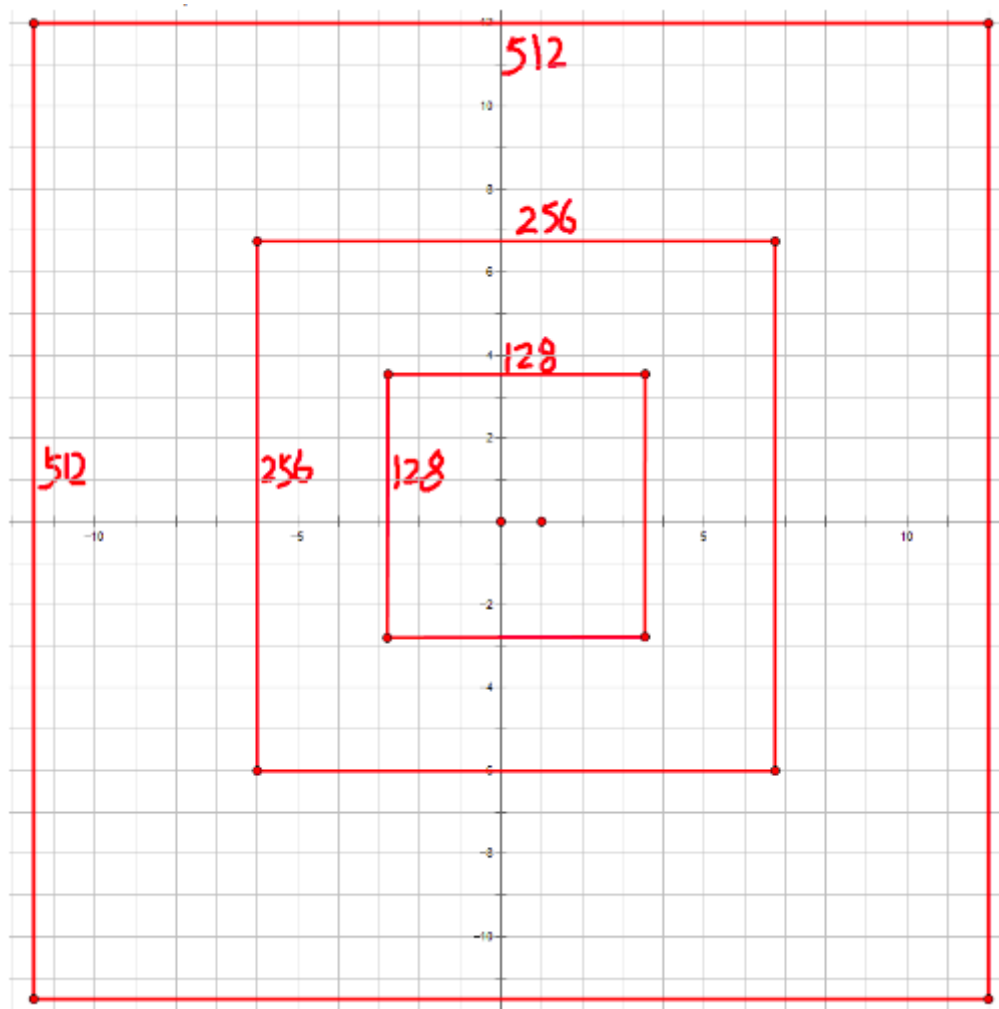
首先 setup部分，

```
1 anchor_scales = layer_params.get('scales', (8, 16, 32))
2 self._anchors = generate_anchors(scales=np.array(anchor_scales))
```

调用generate_anchors方法生成最初始的9个anchor该函数位于generate_anchors.py 主要功能是生成多尺度，多宽高比的anchors，8,16,32其实就是scales:[2^3 2^4 2^5]，base_size为16，具体是怎么实现的可以查阅源代码。_ratio_enum()部分生成三种宽高比 1:2,1:1,2:1的anchor如下图所示：（以下参考 另外一篇博客 (<http://blog.csdn.net/xzzppp/article/details/52317863>)）



_scale_enum()部分，生成三种尺寸的anchor，以_ratio_enum()部分生成的anchor[0 0 15 15]为例，扩展了三种尺度128*128,256*256,512*512，如下图所示：



另外一个函数就是forward()。

在faster rcnn中会根据不同图的输入，得到不同的feature map， $height, width = bottom[0].data.shape[-2:]$ 首先得到conv5的高宽，以及gt box $gt_boxes = bottom[1].data$ ，图片信息 $im_info = bottom[2].data[0, :]$ ，然后计算偏移量， $shift_x =$

`np.arange(0, width) * self._feat_stride`，在这里，你会发现，例如你得到的fm是H=61，W=36，然后你乘以16，得到的图形大概就是1000*600，其实这个16大概就是网络的缩放比例。接下来就是生成anchor，以及对anchor做一定的筛选，详见代码。

另外一个需要理解的就是proposal layer，这个只是在测试的时候用，许多东西和AnchorTargetLayer类似，不详细介绍，可以查看代码。主要看看forward函数，函数算法介绍在注释部分写的很详细：

```
1  # Algorithm:
2  # for each (H, W) location i
3  #   generate A anchor boxes centered on cell i
4  #   apply predicted bbox deltas at cell i to each of the A anchors
5  # clip predicted boxes to image
6  # remove predicted boxes with either height or width < threshold
7  # sort all (proposal, score) pairs by score from highest to lowest
8  # take top pre_nms_topN proposals before NMS
9  # apply NMS with threshold 0.7 to remaining proposals
10 # take after_nms_topN proposals after NMS
11 # return the top proposals (-> RoIs top, scores top)
```

在这个函数中会引用NMS方法。

代码文件夹说明

转载出处：另外一篇博客 (<http://blog.csdn.net/bailufeiyan/article/details/50749694>)

tools

在tools文件夹中，是我们直接调用的最外层的封装文件。其中主要包含的文件为：

`_init_paths.py`：用来初始化路径的，也就是之后的路径会join (path, *)

`compress_net.py`：用来压缩参数的，使用了SVD来进行压缩，这里可以发现，作者对于fc6层和fc7层进行了压缩，也就是两个全连接层。

demo.py : 通常, 我们会直接调用这个函数, 如果要测试自己的模型和数据, 这里需要修改。这里调用了fast_rcnn中的test、config、nums_wrapper函数。vis_detections用来做检测, parse_args用来进行参数设置, 以及damo和主函数。

eval_recall.py : 评估函数

reval.py : re-evaluate, 这里调用了fast_rcnn以及dataset中的函数。其中, from_mats函数和from_dets函数分别loadmat文件和pkl文件。

rpn_genetate.py : 这个函数调用了rpn中的genetate函数, 之后我们会对rpn层做具体的介绍。这里, 主要是一个封装调用的过程, 我们在这里调用配置的参数、设置rpn的test参数, 以及输入输出等操作。

test_net.py : 测试fast rcnn网络。主要就是一些参数配置。

train_faster_rcnn_alt_opt.py : 训练faster rcnn网络使用交替的训练, 这里就是根据faster rcnn文章中的具体实现。可以在主函数中看到, 其包括的步骤为:

RPN 1, 使用imagenet model进行初始化参数, 生成proposal, 这里存储在mp_kwargs

fast rcnn 1, 使用 imagenet model 进行初始化参数, 使用刚刚生成的proposal进行fast rcnn的训练

RPN 2使用 fast rcnn 中的参数进行初始化 (这里要注意哦), 并生成proposal

fast rcnn 2, 使用RPN 2 中的 model进行初始化参数

值得注意的是: 在我们训练时, 我们可以在get_solvers中的max_iters中设置迭代次数, 在不确定网络是否可以调通时, 减少迭代次数可以减少测试时间。

我们在训练faster rcnn网络时, 就是调用这个文件训练的

train_net.py : 使用fast rcnn, 训练自己数据集的网络模型

train_svms.py : 使用最原始的RCNN网络训练post-hoc SVMs

RPN

这里我们主要看lib/rpn文件夹下的代码。这里主要介绍了rpn的模型, 其中, 包含的主要文件如下:

generate_anchors.py: 生成多尺度和多比例的锚点。这里由generate_anchors函数主要完成, 可以看到, 使用了 3 个尺度(128, 256, and 512)以及 3 个比例(1:1,1:2,2:1)。一个锚点由w, h, x_ctr, y_ctr固定, 也就是宽、高、x center和y center固定。

proposal_layer.py:这个函数是用来将RPN的输出转变为object proposals的。作者新增了ProposalLayer类, 这个类中, 重新了set_up和forward函数, 其中forward实现了: 生成锚点box、对于每个锚点提供box的参数细节、将预测框切成图像、删除宽、高小于阈值的框、将所有的(proposal, score) 对排序、获取 pre_nms_topN proposals、获取NMS、获取 after_nms_topN proposals。(注: NMS, nonmaximum suppression, 非极大值抑制)

anchor_target_layer.py:生成每个锚点的训练目标和标签，将其分类为1 (object), 0 (not object) , -1 (ignore).当label>0，也就是有object时，将会进行box的回归。其中，forward函数功能：在每一个cell中，生成9个锚点，提供这9个锚点的细节信息，过滤掉超过图像的锚点，测量同GT的overlap。

proposal_target_layer.py:对于每一个object proposal 生成训练的目标和标签，分类标签从0-k，对于标签>0的box进行回归。（注意，同anchor_target_layer.py不同，两者一个是生成anchor，一个是生成proposal）

generate.py:使用一个rpn生成object proposals。

作者就是通过以上这些文件生成rpn的。

nms

lib/nms文件夹下是非极大值抑制，这部分大家应该已经非常熟悉了，其Python版本的核心函数为py_cpu_nms.py，具体实现以及注释如下：


```
1 def py_cpu_nms(dets, thresh):
2     """Pure Python NMS baseline."""
3     #x1、y1、x2、y2、以及score赋值
4     x1 = dets[:, 0]
5     y1 = dets[:, 1]
6     x2 = dets[:, 2]
7     y2 = dets[:, 3]
8     scores = dets[:, 4]
9
10    #每一个op的面积
11    areas = (x2 - x1 + 1) * (y2 - y1 + 1)
12    #order是按照score排序的
13    order = scores.argsort()[::-1]
14
15    keep = []
16    while order.size > 0:
17        i = order[0]
18        keep.append(i)
19        xx1 = np.maximum(x1[i], x1[order[1:]])
20        yy1 = np.maximum(y1[i], y1[order[1:]])
21        xx2 = np.minimum(x2[i], x2[order[1:]])
22        yy2 = np.minimum(y2[i], y2[order[1:]])
23
24        #计算相交的面积
25        w = np.maximum(0.0, xx2 - xx1 + 1)
26        h = np.maximum(0.0, yy2 - yy1 + 1)
27        inter = w * h
28        #计算：重叠面积/（面积1+面积2-重叠面积）
29        ovr = inter / (areas[i] + areas[order[1:]] - inter)
30
31        inds = np.where(ovr <= thresh)[0]
32        order = order[inds + 1]
```

参考

在这里，没有贴出代码的注释，只是梳理了下Faster RCNN训练的流程，因为代码的注释网络上已经有很多，需要看代码的注释可以参考下面几个博客，我看代码的时候也有参考：

[1] <http://www.cnblogs.com/CarryPotMan/p/5390336.html> (<http://www.cnblogs.com/CarryPotMan/p/5390336.html>)

[2] <http://blog.csdn.net/u010668907/article/category/6237110> (<http://blog.csdn.net/u010668907/article/category/6237110>)

[3] <http://blog.csdn.net/sunyiyou9/article/category/6269359> (<http://blog.csdn.net/sunyiyou9/article/category/6269359>)

[4] <http://blog.csdn.net/bailufeiyan/article/details/50749694> (<http://blog.csdn.net/bailufeiyan/article/details/50749694>)

原文地址：

<http://blog.csdn.net/u011956147/article/details/53053381> (<http://blog.csdn.net/u011956147/article/details/53053381>)

[查看原文>> \(http://blog.csdn.net/u011956147/article/details/53053381\)](http://blog.csdn.net/u011956147/article/details/53053381)



4

看过本文的人也看了：

- 深度学习知识结构图
(<http://lib.csdn.net/base/deeplearning/structure>)
- 物体检测-从RCNN到YOLO
(<http://lib.csdn.net/article/deeplearning/43036>)

- 以resnet作为前置网络的ssd目标提取检测...
(<http://lib.csdn.net/article/deeplearning/52557>)
- 使用自己的数据集训练faster-rcnn
(<http://lib.csdn.net/article/deeplearning/61648>)

- 在Mac下，深度学习CNN库Overfeat的配置...
(<http://lib.csdn.net/article/deeplearning/50581>)

- Ubuntu上配置caffe+SSD及demo演示 (附...
(<http://lib.csdn.net/article/deeplearning/61656>)

发表评论

输入评论内容

发表

22个评论



(http://my.csdn.net/qq_33202928)

qq_33202928 (http://my.csdn.net/qq_33202928)

楼主，我的qq是407968564,相加你好友问你几个问题，我找不到feat_stride的初始值的位置

2017-05-19 17:50:57

回复



(<http://my.csdn.net/u011956147>)

u011956147 (<http://my.csdn.net/u011956147>)

回复qq_33202928：在prototxt中设置

2017-05-20 09:32:07

回复



(http://my.csdn.net/qq_33202928)

qq_33202928 (http://my.csdn.net/qq_33202928)

回复u011956147：楼主，请问具体的文件名字是哪个？在研究你写的东西，很有帮助

2017-05-20 10:05:54

回复



(http://my.csdn.net/qq_33202928)

qq_33202928 (http://my.csdn.net/qq_33202928)

回复u011956147：多谢指点，我挨个找了一下，都找到了，路转粉

2017-05-20 10:22:53

回复



(<http://my.csdn.net/chenjiehua123456789>)

chenjiehua123456789 (<http://my.csdn.net/chenjiehua123456789>)

请问博主为什么我在Netscope生成的网络模型图和你的不一样呢，还提示Can't infer network data shapes. Can't infer output shape of the 'input-data' layer of type 'Python'. Unsupported layer type: 'Python'.警告

2017-04-28 09:45:00


回复

加载更多

联系方式 (<http://www.csdn.net/company/contact.html>) | 版权声明 (<http://www.csdn.net/company/statement.html>) | 法律顾问 (<http://www.csdn.net/company/layer.html>) | 问题报告 (<mailto:webmaster@csdn.net>) | 合作伙伴 (<http://www.csdn.net/friendlink.html>) | 论坛反馈 (<http://bbs.csdn.net/forums/Service>)

网站客服 杂志客服 (<http://wpa.qq.com/msg?d=1&uin=2251809102&site=qq&menu=yes>) 微博客服 (<http://e.weibo.com/csdnsupport/profile>) webmaster@csdn.net (<mailto:webmaster@csdn.net>)

400-660-0108 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 | 江苏乐知网络技术有限公司

京 ICP 证 09002463 号 | Copyright © 1999-2016, CSDN.NET, All Rights Reserved  (<http://www.hd315.gov.cn/beian/view.asp?bianhao=010202001032100010>)