# 一 image classification

1. 训练

|  |
| --- |
| python train\_image\_classifier.py --train\_dir=satellite/train\_dir --dataset\_name=satellite --dataset\_split\_name=train --dataset\_dir=satellite/data --model\_name=inception\_v3 --checkpoint\_path=satellite/pretrained/inception\_v3.ckpt --checkpoint\_exclude\_scopes=InceptionV3/Logits,InceptionV3/AuxLogits --trainable\_scopes=InceptionV3/Logits,InceptionV3/AuxLogits --max\_number\_of\_steps=100000 --batch\_size=32 --learning\_rate=0.001 --learning\_rate\_decay\_type=fixed --save\_interval\_secs=300 --save\_summaries\_secs=2 --log\_every\_n\_steps=10 --optimizer=rmsprop --weight\_decay=0.00004 |

|  |
| --- |
| python eval\_image\_classifier.py --checkpoint\_path=satellite/train\_dir --eval\_dir=satellite/eval\_dir --dataset\_name=satellite --dataset\_split\_name=validation --dataset\_dir=satellite/data --model\_name=inception\_v3 |

# 二 object detection

第一步, 准备数据集.

下载:

|  |
| --- |
| wget http://host.robots.ox.ac.uk/pascal/VOC/voc2012/VOCtrainval\_11-May-2012.tar |

用voc2012.整理成tfrecord模型,以便tf可以识别使用.

|  |
| --- |
| python create\_pascal\_tf\_record.py --data\_dir voc/VOCdevkit/ --year=VOC2012 --set=train --output\_path=voc/pascal\_train.record  python create\_pascal\_tf\_record.py --data\_dir voc/VOCdevkit/ --year=VOC2012 --set=val --output\_path=voc/pascal\_val.record |

拷贝label, 这个是human 可读的label描述标签.

|  |
| --- |
| cp data/pascal\_label\_map.pbtxt voc/ |

在voc中新建 pretrain, 构建pretrain模型.

|  |
| --- |
| wget http://download.tensorflow.org/models/object\_detection/faster\_rcnn\_inception\_resnet\_v2\_atrous\_coco\_11\_06\_2017.tar.gz |

第二步,训练

|  |
| --- |
| **export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim**  chmod a+x ../research/bin/protoc  ../research/bin/protoc object\_detection/protos/\*.proto --python\_out=.  /home/julyedu\_433249/work/tf\_base/research/bin/protoc object\_detection/protos/\*.proto --python\_out=. |

|  |
| --- |
| python train.py --train\_dir voc/train\_dir/ --pipeline\_config\_path voc/voc.config |

第三步, 保存模型.

|  |
| --- |
| python3 export\_inference\_graph.py --input\_type image\_tensor --pipeline\_config\_path voc/voc.config --trained\_checkpoint\_prefix voc/train\_dir/model.ckpt-48 --output\_directory voc/export |

导出的模型是voc/export/frozen\_inference\_graph.pb 文件。

# 三 tensorflow训练过程中对变量值的打印的方法

目的是在训练过程中打印变量的值,不仅仅是打印tensor的情况.

一般使用print打印某个tensor时候.只能打印该tensor确定的shape信息,而且有些shape的维度和输入的图像有关系,会显示出?号的不确定含义.

如何能够在训练过程中实时打印某些变量的值呢?

第一步, 在根目录下构建一个文件tfprint.py.

|  |
| --- |
| # tf print using var  import tensorflow as tf  **tfp\_similarity\_matrix = tf.placeholder(tf.float32)**  # 需要定义一个tensor. |

第二步,在train函数中.

|  |
| --- |
| --- a/dl\_object\_detection/object\_detection/trainer.py  +++ b/dl\_object\_detection/object\_detection/trainer.py  @@ -31,7 +31,7 @@ from object\_detection.core import standard\_fields as fields  from object\_detection.utils import ops as util\_ops  from object\_detection.utils import variables\_helper  from deployment import model\_deploy  -  **+from object\_detection import tfprint**  slim = tf.contrib.slim  @@ -297,7 +297,7 @@ def train(create\_tensor\_dict\_fn, create\_model\_fn, train\_config, master, task,  slim.learning.train(  - train\_tensor,  **+ [train\_tensor,tfprint.tfp\_similarity\_matrix], # 添加训练时要操作的tensor对象.**  logdir=train\_dir,  master=master,  is\_chief=is\_chief, |

第三步,在待打印的变量的文件中.添加如下.

|  |
| --- |
| from object\_detection import tfprint  …  def \_match(self, similarity\_matrix):    **tfprint.tfp\_similarity\_matrix = tf.Print(similarity\_matrix,["argmax\_matcher's input : similarity\_matrix\n",similarity\_matrix],message="[trainning info]")** |

summarize=

是显示多少个单位的数据.默认是3个.

# 四 实战: 训练到tflite部署

## 4.1 预训练模型

<https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md>

里面有个***COCO-trained models***列表

wget <http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v2_coco_2018_03_29.tar.gz>

## 4.2 转换成tflite的

|  |
| --- |
| toco --graph\_def\_file=voc/export/frozen\_inference\_graph.pb --output\_file=voc/export/frozen\_inference\_graph.lite --input\_format=TENSORFLOW\_GRAPHDEF --output\_format=TFLITE --input\_shape=1,299,299,3 --input\_array=image\_tensor --output\_array=detection\_boxes,detection\_scores,detection\_classes,num\_detections --inference\_type=FLOAT --input\_data\_type=FLOAT  freeze\_graph --input\_graph=/tmp/mobilenet\_v1\_224.pb --input\_checkpoint=/tmp/checkpoints/mobilenet-10202.ckpt --input\_binary=true  --output\_graph=/tmp/frozen\_mobilenet\_v1\_224.pb  --output\_node\_names=MobileNet/Predictions/Reshape\_1            python export\_inference\_graph.py --input\_type image\_tensor --pipeline\_config\_path voc/voc.config --trained\_checkpoint\_prefix voc/train\_dir/model.ckpt-1582 --output\_directory voc/export  python export\_inference\_graph.py --input\_type image\_tensor --pipeline\_config\_path voc/mob.config --trained\_checkpoint\_prefix voc/mob\_train\_dir/model.ckpt-0 --output\_directory voc/mob\_export |

网络的输入是: image\_tensor

输出是:

[\_export\_inference\_graph] output\_node\_names:%s **detection\_boxes,detection\_scores,detection\_classes,num\_detections**

Ssd

|  |
| --- |
| # for model in \  ssd\_mobilenet\_v1\_coco\_11\_06\_2017 \  ssd\_inception\_v2\_coco\_11\_06\_2017 \  rfcn\_resnet101\_coco\_11\_06\_2017 \  faster\_rcnn\_resnet101\_coco\_11\_06\_2017 \  faster\_rcnn\_inception\_resnet\_v2\_atrous\_coco\_11\_06\_2017  do \  curl -OL http://download.tensorflow.org/models/object\_detection/$model.tar.gz  tar -xzf $model.tar.gz $model/frozen\_inference\_graph.pb  cp -a $model /opt/graph\_def/ |

curl -OL <http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v1_coco_11_06_2017.tar.gz>

tar -xzf ssd\_mobilenet\_v1\_coco\_11\_06\_2017.tar.gz

python train.py --train\_dir voc/mob\_train\_dir/ --pipeline\_config\_path voc/mob.config

**objectdetection 本身就有一个ssd转tflite的工具.**

python export\_tflite\_ssd\_graph.py --pipeline\_config\_path=voc/mob.config --trained\_checkpoint\_prefix=voc/mob\_train\_dir/model.ckpt-0 --output\_directory=voc/mob\_export --add\_postprocessing\_op=true

python model\_main.py --model\_dir voc/mob\_train\_dir/ --pipeline\_config\_path voc/mob.config

python object\_detection/model\_main.py --model\_dir object\_detection/voc/train\_dir/ --pipeline\_config\_path object\_detection/voc/voc.config

有一个legecy的 train

python legacy/train.py --train\_dir vvv/trainout/ --pipeline\_config\_path vvv/vv.config

detection\_boxes

toco --graph\_def\_file=voc/export/frozen\_inference\_graph.pb --output\_file=voc/export/frozen\_inference\_graph.lite --input\_format=TENSORFLOW\_GRAPHDEF --output\_format=TFLITE --input\_shape=1,299,299,3 --input\_array=image\_tensor --output\_array= box\_encodings --inference\_type=FLOAT --input\_data\_type=FLOAT

可用的toco

|  |
| --- |
| toco --graph\_def\_file=vvv/export/tflite\_graph.pb--output\_file=vvv/litedir/li.lite --input\_format=TENSORFLOW\_GRAPHDEF --output\_format=TFLITE --input\_shape=1,299,299,3 --input\_array=image\_tensor --output\_array='TFLite\_Detection\_PostProcess','TFLite\_Detection\_PostProcess:1','TFLite\_Detection\_PostProcess:2','TFLite\_Detection\_PostProcess:3' --inference\_type=FLOAT --input\_data\_type=FLOAT --allow\_custom\_ops  ~~toco --graph\_def\_file=vvv/export/tflite\_graph.pb--output\_file=vvv/litedir/lili.lite -- --input\_shape=1,300,300,3 --input\_array= normalized\_input\_image\_tensor --output\_array='TFLite\_Detection\_PostProcess','TFLite\_Detection\_PostProcess:1','TFLite\_Detection\_PostProcess:2','TFLite\_Detection\_PostProcess:3' --inference\_type=FLOAT --allow\_custom\_ops~~  toco --graph\_def\_file=vvv/export/tflite\_graph.pb --output\_file=vvv/litedir/lili.lite --input\_shapes=1,**300,300**,3 --input\_arrays=**normalized\_input\_image\_tensor** --output\_arrays=**'TFLite\_Detection\_PostProcess','TFLite\_Detection\_PostProcess:1','TFLite\_Detection\_PostProcess:2','TFLite\_Detection\_PostProcess:3'** --inference\_type=FLOAT --allow\_custom\_ops  toco --input\_file=$OUTPUT\_DIR/tflite\_graph.pb --output\_file=$OUTPUT\_DIR/detect.tflite \  --input\_shapes=1,300,300,3 \  --input\_arrays=normalized\_input\_image\_tensor \  --output\_arrays='TFLite\_Detection\_PostProcess','TFLite\_Detection\_PostProcess:1','TFLite\_Detection\_PostProcess:2','TFLite\_Detection\_PostProcess:3' \  --inference\_type=FLOAT \  --allow\_custom\_ops |

参考:

<https://jefby.github.io/2018/08/20/%E5%B0%86mobilenet-ssd-tensorflow-pb%E8%BD%AC%E6%8D%A2%E4%B8%BAtflite%E7%9A%84%E8%AF%A6%E7%BB%86%E6%AD%A5%E9%AA%A4/>

关键是output\_array是怎么找到的?

## 4.3 训练ssd v2 模型.

第一, 下载v2 预训练模型.

|  |
| --- |
| wget <http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v2_coco_2018_03_29.tar.gz> |

第二, copy configs,并修改:

|  |
| --- |
| 9c9  < num\_classes: 90  ---  > num\_classes: 20  156c156  < fine\_tune\_checkpoint: "PATH\_TO\_BE\_CONFIGURED/model.ckpt"  ---  > fine\_tune\_checkpoint: "vvv/v2\_pretrain/model.ckpt"  175c175  < input\_path: "PATH\_TO\_BE\_CONFIGURED/mscoco\_train.record-?????-of-00100"  ---  > input\_path: "vvv/pascal\_train.record"  177c177  < label\_map\_path: "PATH\_TO\_BE\_CONFIGURED/mscoco\_label\_map.pbtxt"  ---  > label\_map\_path: "vvv/pascal\_label\_map.pbtxt"  189c189  < input\_path: "PATH\_TO\_BE\_CONFIGURED/mscoco\_val.record-?????-of-00010"  ---  > input\_path: "vvv/pascal\_val.record"  191c191  < label\_map\_path: "PATH\_TO\_BE\_CONFIGURED/mscoco\_label\_map.pbtxt"  ---  > label\_map\_path: "vvv/pascal\_label\_map.pbtxt" |

第三, 训练

|  |
| --- |
| python legacy/train.py --train\_dir vvv/trainout/ --pipeline\_config\_path vvv/vv.config |

第四, 转成tflite可用的pb文件.

|  |
| --- |
| python export\_tflite\_ssd\_graph.py --pipeline\_config\_path=voc/mob.config --trained\_checkpoint\_prefix=voc/mob\_train\_dir/model.ckpt-0 --output\_directory=voc/mob\_export --add\_postprocessing\_op=true |

第五, 把pb文件转成tflite文件

|  |
| --- |
| toco --graph\_def\_file=vvv/v2\_export/tflite\_graph.pb --output\_file=vvv/v2\_tflite/thedemo.tflite --input\_shapes=1,300,300,3 --input\_arrays=normalized\_input\_image\_tensor --output\_arrays='TFLite\_Detection\_PostProcess','TFLite\_Detection\_PostProcess:1','TFLite\_Detection\_PostProcess:2','TFLite\_Detection\_PostProcess:3' --inference\_type=FLOAT --allow\_custom\_ops |

4.5 Android端apk部署.

第一,下载最新tensorflow代码

|  |
| --- |
| Git clone https://github.com/tensorflow/tensorflow.git |

第二,找到如下目录的android工程.用as打开.

tensorflow/tensorflow/examples

最新版已经不是这个目录了,还没找到最新版的目录.

# 五 解析ssd的多尺度features maps

## 5.1a 添加flags区分train和analysis

|  |
| --- |
| python legacy/train.py --train\_dir vvv/traindir/ --pipeline\_config\_path vvv/v2mob.config --analysising **true** |

## 5.1 ssd 采用的box predictor为:

|  |
| --- |
| [ConvolutionalBoxPredictor] \_min\_depth: 0  [ConvolutionalBoxPredictor] \_max\_depth: 0  [ConvolutionalBoxPredictor] \_conv\_hyperparams\_fn: <function build.<locals>.scope\_fn at 0x7f7cad310d90>  [ConvolutionalBoxPredictor] \_num\_layers\_before\_predictor: 0  [SSDMetaArch] \_box\_predictor: <object\_detection.predictors.convolutional\_box\_predictor.**ConvolutionalBoxPredictor** object at 0x7f7cad314630>  [ConvolutionalBoxPredictor.\_predict] sorted\_keys ['**box\_encodings**', 'class\_predictions\_with\_background']  [ConvolutionalBoxPredictor.\_predict] head\_name **box\_encodings**  [ConvolutionalBoxPredictor.\_predict] head\_obj <object\_detection.**predictors.heads.box\_head.ConvolutionalBoxHead** object at 0x7f31102d52b0>  [ConvolutionalBoxPredictor.\_predict] head\_name **class\_predictions\_with\_background**  [ConvolutionalBoxPredictor.\_predict] head\_obj <object\_detection.predictors.heads.class\_head.ConvolutionalClassHead object at 0x7f31102d52e8> |

Ssd的预测prediction,区分于分类和回归.



## 5.2 打印关心的变量

在ssd中第一个featuremap之后是一个尺度的卷积输出.想在此处添加一个roi.

那么这个roi的输入需要是**[24 19 19 576],**输出需要是两种,其一是回归**24 1083 1 4**,其二是分类**24 1083 21**,

|  |
| --- |
| if(idx==0):  ## add rfcn roi  tfprint.ssd\_fmap0 = tf.Print(image\_feature,["ssd\_fmap0",tf.shape(image\_feature)],summarize=64)  打印的结果是:  **[ssd\_fmap0][24 19 19 576]**  说明,给ssd的第一张feature map是24, 19x19, 576的.  **[batch\_size, height\_i, width\_i, channels\_i]**  看输出:  **[24 1083 21] 分类**  **[24 1083 1 4] 回归**  回归的是**24 1083 1 4**的  **[batch\_size, num\_anchors\_i, q, code\_size]**  这个是cls分类的输出.shape是 **[batch\_size, num\_anchors\_i, num\_classes + 1]**  对已ingde**[batch\_size, num\_anchors\_i, q, code\_size]**  **num\_predictions\_per\_location \* self.\_box\_code\_size == 1084??**  **self.\_box\_code\_size == 4**  无论分类还是回归,他们都需要num\_predictions\_per\_location\_list(描述对应于feature map的” spatial location, 空间位置”的框预测结果,box predictions,的个数.)  是个数?是个数  还是框的回归坐标?  对于第一个feature map[24 19 19 576]. 需要经过一个roi.  输出同样的:  分类的 [24 1083 21]  回归的 [24 1083 1 4]  现在着手分析roi.  Roi是基于同个feature map. 对于分类和回归,经过不同的conv,输出chn不同.  然后在经ops.batch\_position\_sensitive\_crop\_regions做的roi算法.  输出  这个算法需要boxes.  其中boxes的格式如下:  [num\_boxes, 4] normalized coordinates `[y1, x1, y2, x2]   1. 需要弄明白的:   Rfcn的\_crop\_size的值.   1. 需要弄明白　tf.squeeze　dim和tf.unstack的关系. 2. 可能需要一个slim.conv 再折算一个1083的anchors. 3. 这个可能有些问题.   因为slim.conv的输出chn并不一定就是1083那个维度的.   1. Ssd的   tf.shape(predictions[BOX\_ENCODINGS][0]) 是什么?  [predictions[BOX\_ENCODINGS]][24 **1083** 1 4]  predictions[BOX\_ENCODINGS][1]应该是:[24,**600**,1,4]   1. Rfcn需要的box的size是:   [num\_boxes, 4]. Each box is specified in normalized coordinates [y1, x1, y2, x2] |

## 5.3 打印rfcn的roi的输入输出.

### 第一,先下载pretrained model.

|  |
| --- |
| wget http://download.tensorflow.org/models/object\_detection/rfcn\_resnet101\_coco\_2018\_01\_28.tar.gz |

### 第二,训练

python legacy/train.py --train\_dir vvv/rfcn\_traindir/ --pipeline\_config\_path vvv/rfcn.config --analysising false --logtostderr.

Win10 上训练

### 第三,分析

打印的结果:

|  |
| --- |
| [RfcnBoxPredictor] conv\_hyperparams\_fn: <function build.<locals>.scope\_fn at 0x7f03982ee378>  [RfcnBoxPredictor] num\_spatial\_bins: [3, 3]  [RfcnBoxPredictor] depth: 1024  [RfcnBoxPredictor] crop\_size: [18, 18]  [RfcnBoxPredictor] box\_code\_size: 4  [RfcnBoxPredictor] conv\_hyperparams\_fn: <function build.<locals>.scope\_fn at 0x7f03984c1598>  [RfcnBoxPredictor] num\_spatial\_bins: [3, 3]  [RfcnBoxPredictor] depth: 1024  [RfcnBoxPredictor] crop\_size: [18, 18]  [RfcnBoxPredictor] box\_code\_size: 4 |

分析

|  |
| --- |
| 输入:  [rfcn roi][1 38 50 189]  [batch\_size, height\_i, width\_i, channels\_i]  分类: [1 38 56 189] 分类的chn小  回归: [1 38 56 720] 回归的chn大些.  输出:  分类: [64 1 21]  是[batch\_size \* num\_boxes, 1, total\_classes]  回归: [64 1 20 4]  是[batch\_size \* num\_boxes, 1, self.num\_classes, self.\_box\_code\_size]  **如果要弄成[24 1083 1 4]**  **需要把num\_boxes设置成1, 然后tf.sequeeze掉dim1,然后expand\_dim把 dim2.**  **Box\_code size不需要变.** |

## 5.4 win上训练ssd的命令

|  |
| --- |
| python legacy/train.py --train\_dir vvv/traindir/ --pipeline\_config\_path vvv/v2mob\_win.config --analysising false --logtostderr. |

# 六 在ssd中添加roi处理

## 6.1 需要完成的函数

Rfcn中roi的函数:

|  |
| --- |
| ops.batch\_position\_sensitive\_crop\_regions 将map计算出回归参数.  需要:  location\_feature\_map, 是Map, 在分类和回归时候使用不同的map.  boxes=proposal\_boxes, 是RPN的预测框  crop\_size=self.\_crop\_size,  num\_spatial\_bins=self.\_num\_spatial\_bins,  global\_pool=True |

回归的map:

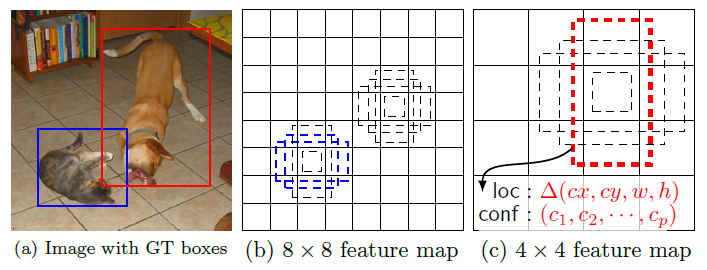
|  |
| --- |
| net = slim.conv2d(net, self.\_depth, [1, 1], scope='reduce\_depth')  # Location predictions.  location\_feature\_map\_depth = (self.\_num\_spatial\_bins[0] \*  self.\_num\_spatial\_bins[1] \*  **self.num\_classes** \*  self.\_box\_code\_size)  location\_feature\_map = slim.conv2d(net, location\_feature\_map\_depth,  [1, 1], activation\_fn=None,  scope='refined\_locations') |

分类的map,和回归使用同一个net,只不过输出的chn要少很多:

|  |
| --- |
| class\_feature\_map\_depth = (self.\_num\_spatial\_bins[0] \*  self.\_num\_spatial\_bins[1] \*  total\_classes)  class\_feature\_map = slim.conv2d(net, class\_feature\_map\_depth, [1, 1],  activation\_fn=None,  scope='class\_predictions') |

6.2 ssd的框预测方法

涉及三个层面:



1. Ssd的prediction,需要image和ground truth框.
2. 在多尺度上做region, 比如在8x8的feature map和4x4的上分别预测框(回归和分类).
3. 然后融合.
4. 这个concat如何做? 依据是什么,在什么dim上做concat?

它有什么问题?

1. 不能有效的处理小物体的识别,其原因是多尺度的feature map的downsize太小了.导致小物体就给忽略了.
2. 如何改进它?
3. 在concat时候添加一个权重,让前层的权重高些.后面的小尺寸的权重低些.
4. 能否添加一个up sample的机制,增大对小物体识别的效果.

有什么应用思路?

1. 在分割领域,识别领域,都可以采用多尺度的featuremap,产生多尺度下的anchors.用以解决识别效率的问题.

# 七 debug roi问题.

## 7.1 ssd的proposal box机制

**a) ssd 的proposal box的尺寸.是一个feature map上产生多个boxes吗?**

[ssd\_fmap0][24 19 19 576][1 24 1083 1 4][1 24 1083 21]

它是 maps regs cls

所谓box部分:

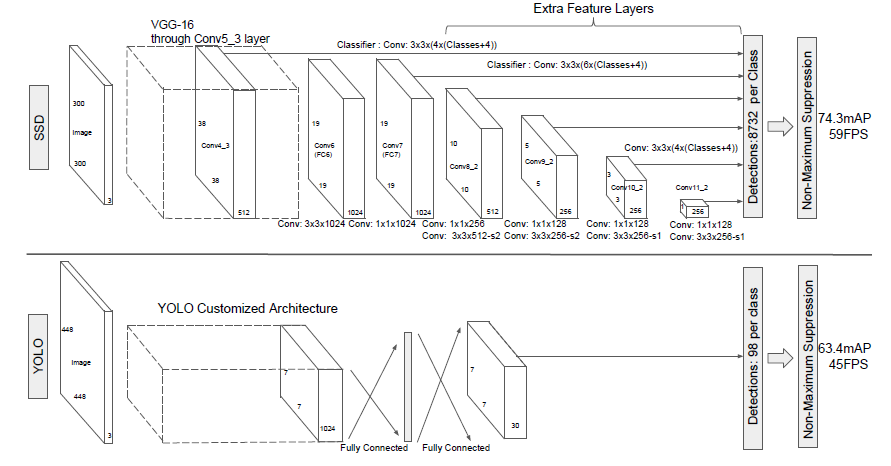
1. 可以认为就是物体框了. 这个也是roi需要的.产生了1083个框.是多个框.

|  |
| --- |
| box\_encodings: A float tensors of shape  [batch\_size, num\_anchors, q, code\_size] representing the location of  the objects, where q is 1 or the number of classes. |

1. 以下论述可以看到是一个featuremap上产生多个proposal boxes.

可以看到feature map是递减的. 输出的anchors也是递减的.batch和chn是不变的.

如图所示



|  |
| --- |
| [ssd\_fmap0,**idx0**,head\_name][box\_encodings][24 19 19 576][24 1083 1 4]  [ssd\_fmap0,**idx0**,head\_name][class\_predictions\_with\_background][24 19 19 576][24 1083 21]  [ssd\_fmap0,idx1,head\_name][class\_predictions\_with\_background][24 10 10 1280][24 600 21]  [ssd\_fmap0,idx2,head\_name][class\_predictions\_with\_background][24 5 5 512][24 150 21]  [ssd\_fmap0,idx3,head\_name][class\_predictions\_with\_background][24 3 3 256][24 54 21]  [ssd\_fmap0,idx4,head\_name][class\_predictions\_with\_background][24 2 2 256][24 24 21]  [ssd\_fmap0,idx5,head\_name][class\_predictions\_with\_background][24 1 1 128][24 6 21] |

1. 一直会有concat不兼容的问题.基本上是batch那个dim不兼容,其他的也不对.需要研究一下这个东西.

首先这个不必要,rfcn是在这里将vgg的**输出降维**用的.我们这里不用吧.

|  |
| --- |
| net\_roi = slim.conv2d(net\_roi, \_depth, [1, 1],reuse=tf.AUTO\_REUSE, scope='reduce\_depth\_roi') |

然后,

打印如下的值:

|  |
| --- |
| batch\_size = tf.shape(proposal\_boxes[0])[0]  num\_boxes = tf.shape(proposal\_boxes[0])[1] |

另外:

Roi的输出:

分类: [64 1 21]

是[batch\_size \* num\_boxes, 1, total\_classes]

回归: [64 1 20 4]

是[batch\_size \* num\_boxes, 1, self.num\_classes, self.\_box\_code\_size]

**如果ssd调用roi的输出要弄成[24 1083 1 4]**

需要把num\_boxes设置成1, 然后tf.sequeeze掉dim1,然后expand\_dim把 dim2.

Box\_code size不需要变.

**试验结果:**

|  |
| --- |
| tfprint.ssd\_debug0 = tf.Print(net\_roi,["reduce depth roi, img, dpt, out; batch\_size,num\_boxes",tf.shape(image\_feature),\_depth,tf.shape(net\_roi),batch\_size,num\_boxes],summarize=8)  [reduce depth roi, img, dpt, out; batch\_size,num\_boxes][24 19 19 576][1024][24 19 19 1024][24][1083] |

分析.

tf.shape(image\_feature) 是 [24 19 19 576]

\_depth 是[1024]

tf.shape(net\_roi) 是 [24 19 19 1024] // 看最后一个chn,从576到1024,这样有一个升维度.

batch\_size 是 [24]

num\_boxes 是 [1083]

1. Proposal尺寸

[proposal\_boxes[0] shape][24][1083][1][4]

b) rfcn的roi代码逻辑. 是**多个maps**对应一个box,分别对**box在maps上**做一个roi结果?

1. 参考8.1和8.2.

简单讲是,同个maps,把depth分成9份.把box分成9份,交替做crop.然后reduce\_mean作为输出. 最后在组成batch个结果.

c) 需要达成的效果:

1. ssd的feature map 0上,划出多个boxes,每个boxes做roi.

d) 查看a,b以便解决c的问题.

#### 7.1.1 ssd 的 sensitive crop region分析

**从下一章,8.1和8.2的分析可以如下的**

###### Pos sen 输出的回归格式

经ops**.**batch\_position\_sensitive\_crop\_regions以及tf.squeeze后的box\_encodings是[24 1083 80]的(**如下面代码片段的黄色高亮**)

|  |
| --- |
| **if(**idx**==**0**):**  ### add roi for 1st feature maps  net\_roi **=** image\_feature  proposal\_boxes **=** predictions**[**BOX\_ENCODINGS**]**  #th slim.arg\_scope(self.\_conv\_hyperparams\_fn()):    \_depth **=** 1024  net\_roi **=** slim**.**conv2d**(**net\_roi**,** \_depth**,** **[**1**,** 1**],**reuse**=**tf**.**AUTO\_REUSE**,** scope**=**'reduce\_depth\_roi'**)**    # Location predictions.  \_num\_spatial\_bins **=** **[**3**,**3**]**  \_num\_classes **=** 20  \_box\_code\_size **=** 4  \_crop\_size **=** **[**18**,** 18**]**  batch\_size **=** tf**.**shape**(**proposal\_boxes**[**0**])[**0**]**  num\_boxes **=** tf**.**shape**(**proposal\_boxes**[**0**])[**1**]**  item2 **=** tf**.**shape**(**proposal\_boxes**[**0**])[**2**]**  item3 **=** tf**.**shape**(**proposal\_boxes**[**0**])[**3**]**  # 这部分的结论已有. 看起来是正确的. net\_roi是[24 19 19 1024]  #tfprint.ssd\_debug0 = tf.Print(net\_roi,["reduce depth roi, img, dpt, out; batch\_size,num\_boxes",tf.shape(image\_feature),\_depth,tf.shape(net\_roi),batch\_size,num\_boxes],summarize=8)  #tfprint.ssd\_debug0 = tf.Print(net\_roi,["proposal\_boxes' shape",batch\_size,num\_boxes,item2,item3],summarize=8)    location\_feature\_map\_depth **=** **(**\_num\_spatial\_bins**[**0**]** **\***  \_num\_spatial\_bins**[**1**]** **\***  \_num\_classes **\***  \_box\_code\_size**)**  location\_feature\_map **=** slim**.**conv2d**(**net\_roi**,** location\_feature\_map\_depth**,**  **[**1**,** 1**],** activation\_fn**=None,**  reuse**=**tf**.**AUTO\_REUSE**,**  scope**=**'refined\_locations\_roi'**)**  ##tf.shape(location\_feature\_map)  proposal\_boxes **=** tf**.**squeeze**(**proposal\_boxes**[**0**],**axis**=[**2**])** #把[24 1083 1 4]的dim0,dim3的"1"挤掉.因为batch\_position\_sensitive\_crop\_regions    box\_encodings **=** ops**.**batch\_position\_sensitive\_crop\_regions**(**  location\_feature\_map**,**  boxes**=**proposal\_boxes**,**  crop\_size**=**\_crop\_size**,**  num\_spatial\_bins**=**\_num\_spatial\_bins**,**  global\_pool**=True)**    box\_encodings **=** tf**.**squeeze**(**box\_encodings**,** squeeze\_dims**=[**2**,** 3**])** #pos reg[24, 1083 1 1 80],带有batch的.  tfprint**.**pos\_sen **=** tf**.**Print**(**image\_feature**,[**"squeezed box"**,**tf**.**shape**(**box\_encodings**)],**summarize**=**8**)##** box\_encodings是[24 1083 80]的  '''注意,如果tf.Print后面接的第一个参数是tensor,如果这个tensor尺寸太大,tf.print会打印它的值.这会导致GPU memory overflow.  建议把tensor设置成一个小值,我们重点看第二列的shape值.''' |

###### Pos sen 输出的分类格式

经batch\_position\_sensitive\_crop\_regions之后的分类结果是**[24 1083 1 1 21]** .

|  |
| --- |
| class\_predictions\_with\_background = (  ops.batch\_position\_sensitive\_crop\_regions(  class\_feature\_map,  boxes=proposal\_boxes,  crop\_size=\_crop\_size,  num\_spatial\_bins=\_num\_spatial\_bins,  global\_pool=True))  '''看一下cls的raw输出.'''  tfprint.pos\_sen = tf.Print(image\_feature,["pos map result of cls",tf.shape(class\_predictions\_with\_background)],summarize=8)## 这里格式是 **[24 1083 1 1 21]**  ''' |

## 7.2 ssd proposal代码实现

在predictor/convolution\_box\_predictor.py中

|  |
| --- |
| def \_predict(self, image\_features, num\_predictions\_per\_location\_list):  """Computes encoded object locations and corresponding confidences.  Args:  **image\_features**: A list of float tensors of shape [batch\_size, height\_i,  width\_i, channels\_i] containing features for a batch of images.  Image的features,这是一组feature maps, 这也是多尺度的体现(multi scopes)  **num\_predictions\_per\_location\_list**: A list of integers representing the  number of box predictions to be made per spatial location for each  feature map.  是list,并且是整数,代表,每一个map的每一个空间位置(spatial location)预测几个框.  Returns:  **box\_encodings**: A list of float tensors of shape  [batch\_size, num\_anchors\_i, q, code\_size] representing the location of  the objects, where q is 1 or the number of classes. Each entry in the  list corresponds to a feature map in the input `image\_features` list.  是list,成员是tensor, 代表物体的location.每一个tensor对应于feature maps之一,是多尺度的结果.  **class\_predictions\_with\_background**: A list of float tensors of shape  [batch\_size, num\_anchors\_i, num\_classes + 1] representing the class  predictions for the proposals. Each entry in the list corresponds to a  feature map in the input `image\_features` list.  是list, 成员是tensor, 代表每个预测框的分类,对应于feature maps之一,是多尺度的结果.  """ |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | predictions = { ## 字典 | |  | BOX\_ENCODINGS: [], | |  | CLASS\_PREDICTIONS\_WITH\_BACKGROUND: [], | |  | } | |  | for head\_name in self.\_other\_heads.keys(): | |  | predictions[head\_name] = [] | |  | # TODO(rathodv): Come up with a better way to generate scope names | |  | # in box predictor once we have time to retrain all models in the zoo. | |  | # The following lines create scope names to be backwards compatible with the | |  | # existing checkpoints. | |  | box\_predictor\_scopes = [\_NoopVariableScope()] | |  | if len(image\_features) > 1: | |  | box\_predictor\_scopes = [ | |  | tf.variable\_scope('BoxPredictor\_{}'.format(i)) | |  | for i in range(len(image\_features)) | |  | ] | |  | for (image\_feature, | |  | num\_predictions\_per\_location, box\_predictor\_scope) in zip( | |  | image\_features, num\_predictions\_per\_location\_list, | |  | box\_predictor\_scopes):  ## 对于每个feature map,以及对应的预测框个数为单位,依次处理所有maps | |  | net = image\_feature # 这是单个map, 从最大开始. | |  | with box\_predictor\_scope: | |  | with slim.arg\_scope(self.\_conv\_hyperparams\_fn()): | |  | with slim.arg\_scope([slim.dropout], is\_training=self.\_is\_training): | |  | # Add additional conv layers before the class predictor. | |  | features\_depth = static\_shape.get\_depth(image\_feature.get\_shape()) | |  | depth = max(min(features\_depth, self.\_max\_depth), self.\_min\_depth) | |  | tf.logging.info('depth of additional conv before box predictor: {}'. | |  | format(depth))  ## 这里的判断条件不满足,在predictor之前没有设置layer,该值可tunning.如果设置了,需要加对应个slim.conv2d,输出的chn是depth的. | |  | if depth > 0 and self.\_num\_layers\_before\_predictor > 0: | |  | for i in range(self.\_num\_layers\_before\_predictor): | |  | net = slim.conv2d( | |  | net, | |  | depth, [1, 1], | |  | reuse=tf.AUTO\_REUSE, | |  | scope='Conv2d\_%d\_1x1\_%d' % (i, depth)) | |  | sorted\_keys = sorted(self.\_other\_heads.keys()) | |  | sorted\_keys.append(BOX\_ENCODINGS) | |  | sorted\_keys.append(CLASS\_PREDICTIONS\_WITH\_BACKGROUND)  ## 对于回归和分类调用不同的子类实现predict | |  | for head\_name in sorted\_keys: | |  | if head\_name == BOX\_ENCODINGS: | |  | head\_obj = self.\_box\_prediction\_head | |  | elif head\_name == CLASS\_PREDICTIONS\_WITH\_BACKGROUND: | |  | head\_obj = self.\_class\_prediction\_head | |  | else: | |  | head\_obj = self.\_other\_heads[head\_name] | |  | prediction = head\_obj.predict( | |  | features=net, | |  | num\_predictions\_per\_location=num\_predictions\_per\_location) | |  | predictions[head\_name].append(prediction) | |  | return predictions | |

### 7.2.1 回归的predict

在predictors/heads/box\_head.py中

|  |
| --- |
| def predict(self, features, num\_predictions\_per\_location):  """Predicts boxes.  Args:  features: A float tensor of shape [batch\_size, height, width, channels]  containing image features.  某一层的feature map  num\_predictions\_per\_location: Number of box predictions to be made per  spatial location. Int specifying number of boxes per location.  该层需要产生的num个预测框.  Returns:  box\_encodings: A float tensors of shape  [batch\_size, num\_anchors, q, code\_size] representing the location of  the objects, where q is 1 or the number of classes.  返回产生的预测框的回归参数.四个参数,w,h,cx,cy(code\_size)  """ |

|  |
| --- |
| net = features  if self.\_use\_depthwise: ## 采用depthwise和pointwise的方法做卷积,可以省计算以及参数量.  box\_encodings = slim.separable\_conv2d(  net, None, [self.\_kernel\_size, self.\_kernel\_size],  padding='SAME', depth\_multiplier=1, stride=1,  rate=1, scope='BoxEncodingPredictor\_depthwise')  box\_encodings = slim.conv2d(  box\_encodings,  num\_predictions\_per\_location \* self.\_box\_code\_size, [1, 1],  activation\_fn=None,  normalizer\_fn=None,  normalizer\_params=None,  scope='BoxEncodingPredictor')  else:  ## fliters是 \_kernel\_size的kernel, 输入是map, 输出是4个位置参数 \* 每个位置需要预测几个预测框  box\_encodings = slim.conv2d(  net, num\_predictions\_per\_location \* self.\_box\_code\_size,  [self.\_kernel\_size, self.\_kernel\_size],  activation\_fn=None,  normalizer\_fn=None,  normalizer\_params=None,  scope='BoxEncodingPredictor')  batch\_size = features.get\_shape().as\_list()[0]  if batch\_size is None:  batch\_size = tf.shape(features)[0]  box\_encodings = tf.reshape(box\_encodings,  [batch\_size, -1, 1, self.\_box\_code\_size])  ## 做一个reshape.  ## 这个reshape的含义是?  ## 1.  return box\_encodings |

### 7.2.2 分类的prediction:

|  |
| --- |
| def predict(self, features, num\_predictions\_per\_location):  """Predicts boxes.  Args:  features: A float tensor of shape [batch\_size, height, width, channels]  containing image features.  # 某一层的map  num\_predictions\_per\_location: Number of box predictions to be made per  spatial location.  # 该层map描述几个预测框  Returns:  class\_predictions\_with\_background: A float tensors of shape  [batch\_size, num\_anchors, num\_classes + 1] representing the class  predictions for the proposals.  描述预测框的分类结果.包含背景.  """ |

|  |
| --- |
| net = features  # Add a slot for the background class.  num\_class\_slots = self.\_num\_classes + 1  if self.\_use\_dropout: ## dropout的机制防止过拟合.  net = slim.dropout(net, keep\_prob=self.\_dropout\_keep\_prob)  if self.\_use\_depthwise: ## depthwise, pointwise机制,减少计算量和参数量.  class\_predictions\_with\_background = slim.separable\_conv2d(  net, None, [self.\_kernel\_size, self.\_kernel\_size],  padding='SAME', depth\_multiplier=1, stride=1,  rate=1, scope='ClassPredictor\_depthwise')  class\_predictions\_with\_background = slim.conv2d(  class\_predictions\_with\_background,  num\_predictions\_per\_location \* num\_class\_slots, [1, 1],  activation\_fn=None,  normalizer\_fn=None,  normalizer\_params=None,  scope='ClassPredictor')  else: ## 将map做一层卷积,输出是每个类对应每个框.  class\_predictions\_with\_background = slim.conv2d(  net,  num\_predictions\_per\_location \* num\_class\_slots,  [self.\_kernel\_size, self.\_kernel\_size],  activation\_fn=None,  normalizer\_fn=None,  normalizer\_params=None,  scope='ClassPredictor',  biases\_initializer=tf.constant\_initializer(  self.\_class\_prediction\_bias\_init))  if self.\_apply\_sigmoid\_to\_scores:  class\_predictions\_with\_background = tf.sigmoid(  class\_predictions\_with\_background)  batch\_size = features.get\_shape().as\_list()[0]  if batch\_size is None:  batch\_size = tf.shape(features)[0]  class\_predictions\_with\_background = tf.reshape( ## reshape结果  class\_predictions\_with\_background, [batch\_size, -1, num\_class\_slots])  return class\_predictions\_with\_background |

## 7.3 ssd的concat代码实现

在meta\_architechtures\ssd\_meta\_arch.py中.

|  |
| --- |
| def predict(self, preprocessed\_inputs, true\_image\_shapes):  """Predicts **unpostprocessed** tensors from input tensor.  未经后处理的预测结果  This function takes an input batch of images and runs it through the forward  pass of the network to yield unpostprocessesed predictions.  A side effect of calling the predict method is that self.\_anchors is  populated with a box\_list.BoxList of anchors. These anchors must be  constructed before the postprocess or loss functions can be called.  注意anchors需要先初始化,才能在此处使用.  Args:  **preprocessed\_inputs**: a [batch, height, width, channels] image tensor.  图像数据  **true\_image\_shapes**: int32 tensor of shape [batch, 3] where each row is  of the form [height, width, channels] indicating the shapes  of true images in the resized images, as resized images can be padded  with zeros.  图像batch对应的shape  比如: [640 480 3]  [300 300 3]  ….  Returns:  **prediction\_dict**: a dictionary holding "raw" prediction tensors:  未经后处理的预测(raw预测)  1) preprocessed\_inputs: the [batch, height, width, channels] image  tensor.  2) box\_encodings: 4-D float tensor of shape [batch\_size, num\_anchors,  box\_code\_dimension] containing predicted boxes.  回顾的尺寸.  3) class\_predictions\_with\_background: 3-D float tensor of shape  [batch\_size, num\_anchors, num\_classes+1] containing class predictions  (logits) for each of the anchors. Note that this tensor \*includes\*  background class predictions (at class index 0).  分类结果.  4) feature\_maps: a list of tensors where the ith tensor has shape  [batch, height\_i, width\_i, depth\_i].  ssd的多尺度的features map.  5) anchors: 2-D float tensor of shape [num\_anchors, 4] containing  the generated anchors in normalized coordinates.  Ssd每个尺度上对应的anchors.(每个anchor是有4个参数的).  """ |

|  |
| --- |
| batchnorm\_updates\_collections **=** **(None** **if** self**.**\_inplace\_batchnorm\_update  **else** tf**.**GraphKeys**.**UPDATE\_OPS**)**  **## 收集更新量**  **if** self**.**\_feature\_extractor**.**is\_keras\_model**:**  feature\_maps **=** self**.**\_feature\_extractor**(**preprocessed\_inputs**)**  **else:**  **with** slim**.**arg\_scope**([**slim**.**batch\_norm**],**  is\_training**=(**self**.**\_is\_training **and**  **not** self**.**\_freeze\_batchnorm**),**  updates\_collections**=**batchnorm\_updates\_collections**):**  **with** tf**.**variable\_scope**(None,** self**.**\_extract\_features\_scope**,**  **[**preprocessed\_inputs**]):**  feature\_maps **=** self**.**\_feature\_extractor**.**extract\_features**(**  preprocessed\_inputs**) ## 利用”mobilenetV2”提取feature maps.**  feature\_map\_spatial\_dims **=** self**.**\_get\_feature\_map\_spatial\_dims**(**  feature\_maps**)**  image\_shape **=** shape\_utils**.**combined\_static\_and\_dynamic\_shape**(**  preprocessed\_inputs**)**  self**.**\_anchors **=** box\_list\_ops**.**concatenate**(**  self**.**\_anchor\_generator**.**generate**(**  feature\_map\_spatial\_dims**,**  im\_height**=**image\_shape**[**1**],**  im\_width**=**image\_shape**[**2**]))## 产生anchors.**  **if** self**.**\_box\_predictor**.**is\_keras\_model**:**  prediction\_dict **=** self**.**\_box\_predictor**(**feature\_maps**)**  **else:**  **with** slim**.**arg\_scope**([**slim**.**batch\_norm**],**  is\_training**=(**self**.**\_is\_training **and**  **not** self**.**\_freeze\_batchnorm**),**  updates\_collections**=**batchnorm\_updates\_collections**):**  prediction\_dict **=** self**.**\_box\_predictor**.**predict**(**  feature\_maps**,** self**.**\_anchor\_generator**.**num\_anchors\_per\_location**()) ## 利用feature maps来做一个未经后处理的raw预测.**  box\_encodings **=** tf**.**concat**(**prediction\_dict**[**'box\_encodings'**],** axis**=**1**)## 将回归结果concat. Dim1是num\_boxes.**  **if** box\_encodings**.**shape**.**ndims **==** 4 **and** box\_encodings**.**shape**[**2**]** **==** 1**:**  box\_encodings **=** tf**.**squeeze**(**box\_encodings**,** axis**=**2**)**  class\_predictions\_with\_background **=** tf**.**concat**(**  prediction\_dict**[**'class\_predictions\_with\_background'**],** axis**=**1**)## 将分类结果进行concat,在dim1上做concat. Dim1是num\_boxes.**  predictions\_dict **=** **{**  'preprocessed\_inputs'**:** preprocessed\_inputs**,**  'box\_encodings'**:** box\_encodings**,**  'class\_predictions\_with\_background'**:**  class\_predictions\_with\_background**,**  'feature\_maps'**:** feature\_maps**,**  'anchors'**:** self**.**\_anchors**.**get**()**  **}**  self**.**\_batched\_prediction\_tensor\_names **=** **[**x **for** x **in** predictions\_dict  **if** x **!=** 'anchors'**]**  **return** predictions\_dict |

###### 问:在什么上concat? 为何引入这样的concat? 解决什么问题?

**答:** 在dim1上(即在num\_boxes上)做的concat.在batch上做concat是说不通的.在chn上也没意义.

**在num\_boxes上concat的物理意义是, 把所有的多尺度的boxes的结果都列在一起.**

比如 第一个feature map的proposal\_box是 [24, 1083, 1,4]的shape

第二个feature map的proposal\_box是[24, 800, 1, 4]的shape.

第三个…

**那结果就是: [24, 1083+800+…, 1, 4]的shape.**

|  |
| --- |
| concat\_dim：0表示行，1表示列  t1 = [[1,2,3], [4,5,6]]  t2 = [[7,8,9], [10,11,12]]  tf.concat(0, [t1, t2]) ==> [[1,2,3], [4,5,6], [7,8,9], [10,11,12]]  tf.concat(1, [t1, t2]) ==> [[1,2,3,7,8, 9], [4,5,6,10,11, 12]]  可以这样思考:  Dim0 对应第一组中括号(以t1为例), [ A, B] ,其中A=[1,2,3],B类似.concat在dim0上,是在第一组中括号上添加, [A,B,C,D]的形式,为[[1,2,3], [4,5,6], [7,8,9], [10,11,12]]  Dim1 对应第二组中括号[[a,b,c],…] 其中a=1,b=2,c=3. Concat在dim1上是在第二组括号中操作,为[[1,2,3,7,8, 9], [4,5,6,10,11, 12]] |

### 7.3.1 concat的shape

|  |
| --- |
| (op: 'ConcatV2') with input shapes: [24,1083,21], [24,1083,21], [24,1083,21], [24,600,21], [24,150,21], [24,54,21], [24,24,21], [24,6,21], |

## 7.4 ssd Loss

加入ssd roi的Loss时候会报错

|  |
| --- |
| Shapes are [24,4083,4] and [24,1917,4]. for 'Loss/Loss/Select' (op: 'Select') with input shapes: [24,1917,4], [24,4083,4], [24,1917,4]  这里是 |

1917 = 1083+600+150+54+24+6

4083 = 1917+1083+1083

为什么会多了两个1083? 是多了reg和cls的.

|  |
| --- |
| loss {  classification\_loss {  weighted\_sigmoid {  }  }  localization\_loss {  **weighted\_smooth\_l1** { **## 采用的loss 函数. SmoothL1**  }  }  hard\_example\_miner {  num\_hard\_examples: 3000  iou\_threshold: 0.99  loss\_type: CLASSIFICATION  max\_negatives\_per\_positive: 3  min\_negatives\_per\_image: 3  }  classification\_weight: 1.0  localization\_weight: 1.0  } |

**smooth L1 loss让loss对于离群点更加鲁棒**，即：相比于L2损失函数，**其对离群点、异常值（outlier）不敏感，梯度变化相对更小，训练时不容易跑飞**。

出现这个问题,可能是预测和target的(groudtruth)的num\_boxes不兼容了.如下图的黄色高亮.

|  |
| --- |
| **class** **Loss(**object**):**  """Abstract base class for loss functions."""  \_\_metaclass\_\_ **=** ABCMeta  **def** \_\_call\_\_**(**self**,**  **prediction\_tensor, ## 预测tensor, [24 4083 4]**  **target\_tensor, ## 目标 tensor [24 1917 4]**  ignore\_nan\_targets**=False,**  scope**=None,**  **\*\***params**):**  """Call the loss function.  Args:  **prediction\_tensor**: an N-d tensor of shape [batch, anchors, ...]  **representing predicted quantities**.  **target\_tensor**: an N-d tensor of shape [batch, anchors, ...] representing  **regression or classification targets**.  ignore\_nan\_targets: whether to ignore nan targets in the loss computation.  E.g. can be used if the target tensor is missing groundtruth data that  shouldn't be factored into the loss.  scope: Op scope name. Defaults to 'Loss' if None.  \*\*params: Additional keyword arguments for specific implementations of  the Loss.  Returns:  loss: a tensor representing the value of the loss function.  """ |

7.4.1 \_\_call\_\_机制

所谓\_\_call\_\_\_, 就是在该类被调用的时候才会先调用.

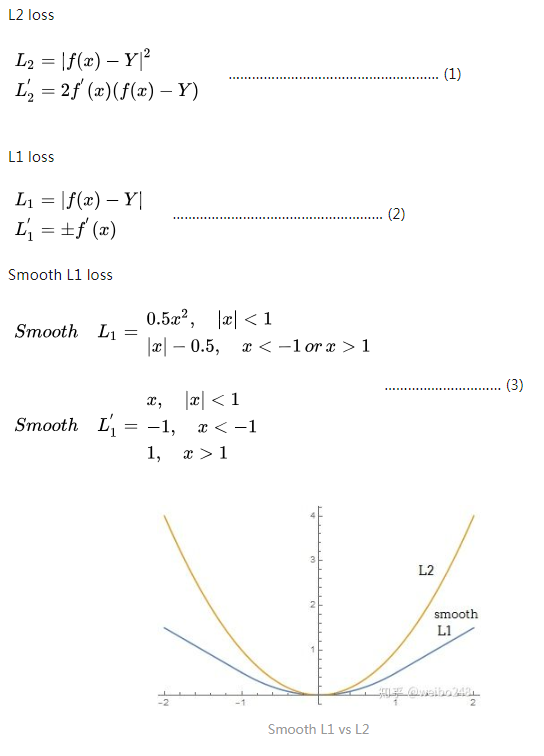
在builders\model\_builder.py中

|  |
| --- |
| **(**classification\_loss**,** localization\_loss**,** classification\_weight**,**  localization\_weight**,** hard\_example\_miner**,**  random\_example\_sampler**)** **=** losses\_builder**.**build**(**ssd\_config**.**loss**)**  ## 利用builder来构建loss  **return** ssd\_meta\_arch**.**SSDMetaArch**(**  is\_training**,**  anchor\_generator**,**  ssd\_box\_predictor**,**  box\_coder**,**  feature\_extractor**,**  matcher**,**  region\_similarity\_calculator**,**  encode\_background\_as\_zeros**,**  negative\_class\_weight**,**  image\_resizer\_fn**,**  non\_max\_suppression\_fn**,**  score\_conversion\_fn**,**  **classification\_loss, ## 配置给ssd model**  **localization\_loss,**  classification\_weight**,**  localization\_weight**,**  normalize\_loss\_by\_num\_matches**,**  hard\_example\_miner**,**  target\_assigner\_instance**=**target\_assigner\_instance**,**  add\_summaries**=**add\_summaries**,**  normalize\_loc\_loss\_by\_codesize**=**normalize\_loc\_loss\_by\_codesize**,**  freeze\_batchnorm**=**ssd\_config**.**freeze\_batchnorm**,**  inplace\_batchnorm\_update**=**ssd\_config**.**inplace\_batchnorm\_update**,**  add\_background\_class**=**add\_background\_class**,**  random\_example\_sampler**=**random\_example\_sampler**,**  expected\_classification\_loss\_under\_sampling**=**  expected\_classification\_loss\_under\_sampling**)** |

|  |
| --- |
| **def** \_build\_localization\_loss**(**loss\_config**):**  ## 依据v2mob.config的loss配置, 构建一个reg时候的loss.  这里是smoothL1的loss.  """Builds a localization loss based on the loss config.  Args:  loss\_config: A losses\_pb2.LocalizationLoss object.  Returns:  Loss based on the config.  Raises:  ValueError: On invalid loss\_config.  """  **if** **not** isinstance**(**loss\_config**,** losses\_pb2**.**LocalizationLoss**):**  **raise** ValueError**(**'loss\_config not of type losses\_pb2.LocalizationLoss.'**)**  loss\_type **=** loss\_config**.**WhichOneof**(**'localization\_loss'**)**  **if** loss\_type **==** 'weighted\_l2'**:**  **return** losses**.**WeightedL2LocalizationLoss**()**  **if** loss\_type **==** 'weighted\_smooth\_l1'**:**  **return** losses**.**WeightedSmoothL1LocalizationLoss**(**  loss\_config**.**weighted\_smooth\_l1**.**delta**) ## 采用这里的loss**  **if** loss\_type **==** 'weighted\_iou'**:**  **return** losses**.**WeightedIOULocalizationLoss**()**  **raise** ValueError**(**'Empty loss config.'**)** |

在如下位置才\_\_call\_\_到.

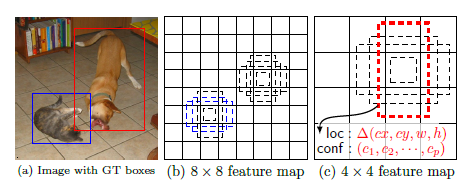
|  |
| --- |
| **def** loss**(**self**,** prediction\_dict**,** true\_image\_shapes**,** scope**=None):**  """Compute scalar loss tensors with respect to provided groundtruth.  Calling this function requires that groundtruth tensors have been  provided via the provide\_groundtruth function.  Args:  prediction\_dict: a dictionary holding prediction tensors with  1) box\_encodings: 3-D float tensor of shape [batch\_size, num\_anchors,  box\_code\_dimension] containing predicted boxes.  2) class\_predictions\_with\_background: 3-D float tensor of shape  [batch\_size, num\_anchors, num\_classes+1] containing class predictions  (logits) for each of the anchors. Note that this tensor \*includes\*  background class predictions.  true\_image\_shapes: int32 tensor of shape [batch, 3] where each row is  of the form [height, width, channels] indicating the shapes  of true images in the resized images, as resized images can be padded  with zeros.  scope: Optional scope name.  Returns:  a dictionary mapping loss keys (`localization\_loss` and  `classification\_loss`) to scalar tensors representing corresponding loss  values.  """  **with** tf**.**name\_scope**(**scope**,** 'Loss'**,** prediction\_dict**.**values**()):**  keypoints **=** **None**  **if** self**.**groundtruth\_has\_field**(**fields**.**BoxListFields**.**keypoints**):**  keypoints **=** self**.**groundtruth\_lists**(**fields**.**BoxListFields**.**keypoints**)**  weights **=** **None**  **if** self**.**groundtruth\_has\_field**(**fields**.**BoxListFields**.**weights**):**  weights **=** self**.**groundtruth\_lists**(**fields**.**BoxListFields**.**weights**)**  **(batch\_cls\_targets,** batch\_cls\_weights**,** **batch\_reg\_targets,**  batch\_reg\_weights**,** match\_list**)** **=** self**.**\_assign\_targets**(**  self**.**groundtruth\_lists**(**fields**.**BoxListFields**.**boxes**),**  self**.**groundtruth\_lists**(**fields**.**BoxListFields**.**classes**),**  keypoints**,** weights**) ## 得到ground truth. 需要添加roi部分的groudtruth.**  **if** self**.**\_add\_summaries**:**  self**.**\_summarize\_target\_assignment**(**  self**.**groundtruth\_lists**(**fields**.**BoxListFields**.**boxes**),** match\_list**)**  **if** self**.**\_random\_example\_sampler**:**  batch\_sampled\_indicator **=** tf**.**to\_float**(**  shape\_utils**.**static\_or\_dynamic\_map\_fn**(**  self**.**\_minibatch\_subsample\_fn**,**  **[**batch\_cls\_targets**,** batch\_cls\_weights**],**  dtype**=**tf**.**bool**,**  parallel\_iterations**=**self**.**\_parallel\_iterations**,**  back\_prop**=True))**  batch\_reg\_weights **=** tf**.**multiply**(**batch\_sampled\_indicator**,**  batch\_reg\_weights**)**  batch\_cls\_weights **=** tf**.**multiply**(**batch\_sampled\_indicator**,**  batch\_cls\_weights**)**  location\_losses **=** **self.\_localization\_loss(**  prediction\_dict**[**'box\_encodings'**], ## 预测框, [24 4083 4]**  batch\_reg\_targets**, ## groundtruth [24 1917 4] , 能否把它提升到和预测框shape一致了?**  ignore\_nan\_targets**=True,**  weights**=**batch\_reg\_weights**)**  cls\_losses **=** **self.\_classification\_loss(**  prediction\_dict**[**'class\_predictions\_with\_background'**],**  batch\_cls\_targets**,**  weights**=**batch\_cls\_weights**)** |



## 7.5 ssd的anchor generator

在anchor\_generator\_builder.py中对ssd的anchor generator做了创建.

|  |
| --- |
| #从config中读取配置, 搭建multiple\_grid\_anchor\_generator  multiple\_grid\_anchor\_generator.create\_ssd\_anchors |



默认anchor如图.

配置表如下:

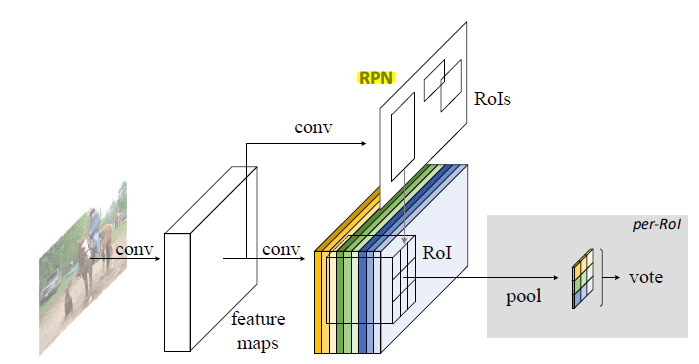
|  |
| --- |
| anchor\_generator {  ssd\_anchor\_generator {  num\_layers: 6 # 对应6层的anchors  min\_scale: 0.2  max\_scale: 0.95  aspect\_ratios: 1.0  aspect\_ratios: 2.0  aspect\_ratios: 0.5  aspect\_ratios: 3.0  aspect\_ratios: 0.3333 ## 比例为5种(宽高比?)  }  } |

|  |
| --- |
| **def** create\_ssd\_anchors**(**num\_layers**=**6**,**  min\_scale**=**0.2**,**  max\_scale**=**0.95**,**  scales**=None,**  aspect\_ratios**=(**1.0**,** 2.0**,** 3.0**,** 1.0 **/** 2**,** 1.0 **/** 3**),**  interpolated\_scale\_aspect\_ratio**=**1.0**,**  base\_anchor\_size**=None,**  anchor\_strides**=None,**  anchor\_offsets**=None,**  reduce\_boxes\_in\_lowest\_layer**=True):**  """Creates MultipleGridAnchorGenerator for SSD anchors.  This function instantiates a MultipleGridAnchorGenerator that reproduces  ``default box`` construction proposed by Liu et al in the SSD paper. ## 实现论文中2.2节的默认anchors  See **Section 2.2** for details. Grid sizes are assumed to be passed in  at generation time from finest resolution to coarsest resolution --- this is  used to (linearly) interpolate scales of anchor boxes corresponding to the  intermediate grid sizes.  Anchors that are returned by calling the `generate` method on the returned ## 由’generate’函数产生anchors  MultipleGridAnchorGenerator object are always in normalized coordinates  and clipped to the unit square: (i.e. all coordinates lie in [0, 1]x[0, 1]).  Args:  num\_layers: integer number of grid layers to create anchors for (actual  grid sizes passed in at generation time)  min\_scale: scale of anchors corresponding to finest resolution (float)  max\_scale: scale of anchors corresponding to coarsest resolution (float)  scales: As list of anchor scales to use. When not None and not empty,  min\_scale and max\_scale are not used. ## min\_scale和max\_scale和scales互斥  aspect\_ratios: list or tuple of (float) aspect ratios to place on each  grid point.  interpolated\_scale\_aspect\_ratio: An additional anchor is added with this  aspect ratio and a scale interpolated between the scale for a layer  and the scale for the next layer (1.0 for the last layer).  This anchor is not included if this value is 0.  base\_anchor\_size: base anchor size as [height, width].  The height and width values are normalized to the minimum dimension of the  input height and width, so that when the base anchor height equals the  base anchor width, the resulting anchor is square even if the input image  is not square.  anchor\_strides: list of pairs of strides in pixels (in y and x directions  respectively). For example, setting anchor\_strides=[(25, 25), (50, 50)]  means that we want the anchors corresponding to the first layer to be  strided by 25 pixels and those in the second layer to be strided by 50  pixels in both y and x directions. If anchor\_strides=None, they are set to  be the reciprocal of the corresponding feature map shapes.  anchor\_offsets: list of pairs of offsets in pixels (in y and x directions  respectively). The offset specifies where we want the center of the  (0, 0)-th anchor to lie for each layer. For example, setting  anchor\_offsets=[(10, 10), (20, 20)]) means that we want the  (0, 0)-th anchor of the first layer to lie at (10, 10) in pixel space  and likewise that we want the (0, 0)-th anchor of the second layer to lie  at (25, 25) in pixel space. If anchor\_offsets=None, then they are set to  be half of the corresponding anchor stride.  reduce\_boxes\_in\_lowest\_layer: a boolean to indicate whether the fixed 3  boxes per location is used in the lowest layer.  Returns:  a MultipleGridAnchorGenerator  """ |

|  |
| --- |
| **if** base\_anchor\_size **is** **None:**  base\_anchor\_size **=** **[**1.0**,** 1.0**]**  base\_anchor\_size **=** tf**.**constant**(**base\_anchor\_size**,** dtype**=**tf**.**float32**) ## 基准anchor size.**  box\_specs\_list **=** **[]**  **if** scales **is** **None** **or** **not** scales**:**  **## 需要给6层layer产生anchors, 每层layer有不同的scale. 他们从min\_scale到max\_scale线性递增(等差数列).**  scales **=** **[**min\_scale **+** **(**max\_scale **-** min\_scale**)** **\*** i **/** **(**num\_layers **-** 1**)**  **for** i **in** range**(**num\_layers**)]** **+** **[**1.0**]**  **else:**  # Add 1.0 to the end, which will only be used in scale\_next below and used  # for computing an interpolated scale for the largest scale in the list.  scales **+=** **[**1.0**]**  **for** layer**,** scale**,** scale\_next **in** zip**(**  range**(**num\_layers**),** scales**[:-**1**],** scales**[**1**:]):**  **## 取出每层的scale和下一层的scale.**  layer\_box\_specs **=** **[]**  **if** layer **==** 0 **and** reduce\_boxes\_in\_lowest\_layer**:**  layer\_box\_specs **=** **[(**0.1**,** 1.0**),** **(**scale**,** 2.0**),** **(**scale**,** 0.5**)]**  **else:**  **for** aspect\_ratio **in** aspect\_ratios**:**  layer\_box\_specs**.**append**((**scale**,** aspect\_ratio**))**  # Add one more anchor, with a scale between the current scale, and the  # scale for the next layer, with a specified aspect ratio (1.0 by  # default).  **if** interpolated\_scale\_aspect\_ratio **>** 0.0**:**  layer\_box\_specs**.**append**((**np**.**sqrt**(**scale**\***scale\_next**),**  interpolated\_scale\_aspect\_ratio**))**  box\_specs\_list**.**append**(**layer\_box\_specs**)** |

# 八 rfcn的roi问题

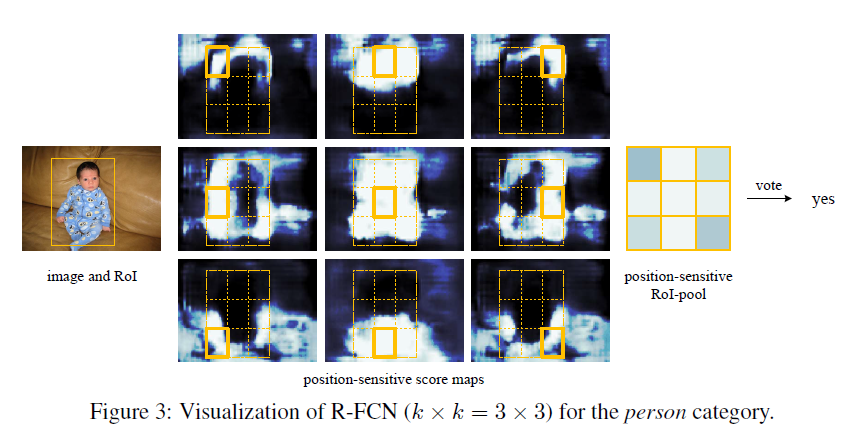
## 8.1 sensitive\_crop\_features的代码实现



|  |
| --- |
| **def** batch\_position\_sensitive\_crop\_regions**(**images**,**  boxes**,**  crop\_size**,**  num\_spatial\_bins**,**  global\_pool**,**  parallel\_iterations**=**64**):**  """Position sensitive crop with batches of images and boxes.  This op is exactly like `position\_sensitive\_crop\_regions` below but operates  on batches of images and boxes. See `position\_sensitive\_crop\_regions` function  below for the operation applied per batch element.  Args:  **images**: A `Tensor`. Must be one of the following types: `uint8`, `int8`,  `int16`, `int32`, `int64`, `half`, `float32`, `float64`.  A 4-D tensor of shape `[batch, image\_height, image\_width, depth]`.  Both `image\_height` and `image\_width` need to be positive.  **boxes**: A `Tensor` of type `float32`.  A 3-D tensor of shape `[batch, num\_boxes, 4]`. Each box is specified in  normalized coordinates `[y1, x1, y2, x2]`. A normalized coordinate value  需要的boxes, 尺寸是[b, n, 4]  of `y` is mapped to the image coordinate at `y \* (image\_height - 1)`, so  as the `[0, 1]` interval of normalized image height is mapped to  `[0, image\_height - 1] in image height coordinates. We do allow y1 > y2,  in which case the sampled crop is an up-down flipped version of the  original image. The width dimension is treated similarly.  crop\_size: See `position\_sensitive\_crop\_regions` below.  num\_spatial\_bins: See `position\_sensitive\_crop\_regions` below.  global\_pool: See `position\_sensitive\_crop\_regions` below.  parallel\_iterations: Number of batch items to process in parallel.  Returns:  """ |

具体实现的代码:

|  |
| --- |
| **def** position\_sensitive\_crop\_regions**(**image**,**  boxes**,**  crop\_size**,**  num\_spatial\_bins**,**  global\_pool**):**  """Position-sensitive crop and pool rectangular regions from a feature grid. 位置敏感矩形框(起到crop和pool作用)  The output crops are split into `spatial\_bins\_y` vertical bins  and `spatial\_bins\_x` horizontal bins. For each intersection of a vertical  and a horizontal bin the output values are gathered by performing  `tf.image.crop\_and\_resize` (bilinear resampling) on a a separate subset of  channels of the image. This reduces `depth` by a factor of  `(spatial\_bins\_y \* spatial\_bins\_x)`.  When global\_pool is True, this function implements a differentiable version  of **position-sensitive RoI** pooling used in  [R-FCN detection system](https://arxiv.org/abs/1605.06409).  **注意,实现的是一个”不同”的版本, 那不同点在?**  When global\_pool is False, this function implements a differentiable version  of position-sensitive assembling operation used in  [instance FCN](https://arxiv.org/abs/1603.08678).  Args:  image: A `Tensor`. Must be one of the following types: `uint8`, `int8`,  `int16`, `int32`, `int64`, `half`, `float32`, `float64`.  A 3-D tensor of shape `[image\_height, image\_width, depth]`.  Both `image\_height` and `image\_width` need to be positive.  boxes: A `Tensor` of type `float32`.  A 2-D tensor of shape `[num\_boxes, 4]`. Each box is specified in  normalized coordinates `[y1, x1, y2, x2]`. A normalized coordinate value  of `y` is mapped to the image coordinate at `y \* (image\_height - 1)`, so  as the `[0, 1]` interval of normalized image height is mapped to  `[0, image\_height - 1] in image height coordinates. We do allow y1 > y2,  in which case the sampled crop is an up-down flipped version of the  original image. The width dimension is treated similarly.  crop\_size: A list of two integers `[crop\_height, crop\_width]`. All  cropped image patches are resized to this size. The aspect ratio of the  image content is not preserved. Both `crop\_height` and `crop\_width` need  to be positive.  num\_spatial\_bins: A list of two integers `[spatial\_bins\_y, spatial\_bins\_x]`.  Represents the number of position-sensitive bins in y and x directions.  Both values should be >= 1. `crop\_height` should be divisible by  `spatial\_bins\_y`, and similarly for width.  The number of image channels should be divisible by  (spatial\_bins\_y \* spatial\_bins\_x).  Suggested value from R-FCN paper: [3, 3].  global\_pool: A boolean variable.  If True, we perform average global pooling on the features assembled from  the position-sensitive score maps.  If False, we keep the position-pooled features without global pooling  over the spatial coordinates.  Note that using global\_pool=True is equivalent to but more efficient than  running the function with global\_pool=False and then performing global  average pooling.  Returns:  position\_sensitive\_features: A 4-D tensor of shape  `[num\_boxes, K, K, crop\_channels]`,  where `**crop\_channels = depth / (spatial\_bins\_y \* spatial\_bins\_x)**`,  where **K = 1 when global\_pool is True** (Average-pooled cropped regions),  and K = crop\_size when global\_pool is False.  返回的是features.  Raises:  ValueError: Raised in four situations:  `num\_spatial\_bins` is not >= 1;  `num\_spatial\_bins` does not divide `crop\_size`;  `(spatial\_bins\_y\*spatial\_bins\_x)` does not divide `depth`;  `bin\_crop\_size` is not square when global\_pool=False due to the  constraint in function space\_to\_depth.  """ |



|  |
| --- |
| total\_bins **=** 1  bin\_crop\_size **=** **[]**  **for** **(**num\_bins**,** crop\_dim**)** **in** zip**(**num\_spatial\_bins**,** crop\_size**):**  **if** num\_bins **<** 1**:**  **raise** ValueError**(**'num\_spatial\_bins should be >= 1'**)**  **if** crop\_dim **%** num\_bins **!=** 0**:**  **raise** ValueError**(**'crop\_size should be divisible by num\_spatial\_bins'**)**  ## 将每个预测框,横竖切出3x3个框(bins).  total\_bins **\*=** num\_bins ## 这个for循环,会计算出总共9个bins  bin\_crop\_size**.**append**(**crop\_dim **//** num\_bins**)**  **if** **not** global\_pool **and** bin\_crop\_size**[**0**]** **!=** bin\_crop\_size**[**1**]:**  **raise** ValueError**(**'Only support square bin crop size for now.'**)**  ymin**,** xmin**,** ymax**,** xmax **=** tf**.**unstack**(**boxes**,** axis**=**1**) ## 解码框的尺寸**  spatial\_bins\_y**,** spatial\_bins\_x **=** num\_spatial\_bins  # Split each box into spatial\_bins\_y \* spatial\_bins\_x bins.  position\_sensitive\_boxes **=** **[]**  **for** bin\_y **in** range**(**spatial\_bins\_y**):**  step\_y **=** **(**ymax **-** ymin**)** **/** spatial\_bins\_y  **for** bin\_x **in** range**(**spatial\_bins\_x**):**  step\_x **=** **(**xmax **-** xmin**)** **/** spatial\_bins\_x  box\_coordinates **=** **[**ymin **+** bin\_y **\*** step\_y**,**  xmin **+** bin\_x **\*** step\_x**,**  ymin **+** **(**bin\_y **+** 1**)** **\*** step\_y**,**  xmin **+** **(**bin\_x **+** 1**)** **\*** step\_x**,**  **] ## 计算出9个bins每个bin的尺寸.**    position\_sensitive\_boxes**.**append**(**tf**.**stack**(**box\_coordinates**,** axis**=**1**)) ## 这9个bin的尺寸参数就组成了”位置敏感”的boxes**    **image\_splits = tf.split(value=image, num\_or\_size\_splits=total\_bins, axis=2) ## 将输入的feature map的dim2切出9个小张量.(区别于9个bins).这个feature map本来是[image\_height, image\_width, depth]尺寸的.意思是在depth上分出9个块.但是本身不会改变shape的个数,当然shape中的数值会减小的.**    image\_crops **=** **[]**  **for** **(**split**,** box**)** **in** zip**(**image\_splits**,** position\_sensitive\_boxes**):**  **if** split**.**shape**.**is\_fully\_defined**()** **and** box**.**shape**.**is\_fully\_defined**():**  crop **=** matmul\_crop\_and\_resize**(**  tf**.**expand\_dims**(**split**,** 0**),** box**,** bin\_crop\_size**)**  **else:**  **## 对于每个小张量和每个bins,做一个crop, 造成一个效果是:**   1. **Feature map的depth0 和左上角的bin做crop.** 2. **Feature map的depth1 和上中的bin做crop.** 3. **Feature map的depth2 和右上角的bin做crop.** 4. **等等..和下图一致,下图共有9个feature map的depth**   **和9个bins. 这样错开的做crop.**    crop **=** **tf.image.crop\_and\_resize(**  tf**.**expand\_dims**(**split**,** 0**),** box**,**  tf**.**zeros**(**tf**.**shape**(**boxes**)[**0**],** dtype**=**tf**.**int32**),** bin\_crop\_size**)**  image\_crops**.**append**(**crop**)**  **if** global\_pool**:**  # Average over all bins.  **## 做一个均值pooling,结果是一个小bin大小的feature maps的数值均值.**  **这里的均值还不涉及chn,这个是多个crops单元的对应维度上的均值.**  position\_sensitive\_features **=** tf**.**add\_n**(**image\_crops**)** **/** len**(**image\_crops**)**  # Then average over spatial positions within the bins.  position\_sensitive\_features **=** **tf.reduce\_mean(**  position\_sensitive\_features**,** **[**1**,** 2**],** keep\_dims**=True)**  **## ~~应该是每个chn上求均值,最后每个batch上求均值.输出一个[h,w]的两维的tensor. 即为返回值.~~**  **~~这个是batch和chn上的加和均值.~~**  Image是[image\_height, image\_width, depth]的    **else:**  # Reorder height/width to depth channel.  block\_size **=** bin\_crop\_size**[**0**]**  **if** block\_size **>=** 2**:**  image\_crops **=** **[**tf**.**space\_to\_depth**(**  crop**,** block\_size**=**block\_size**)** **for** crop **in** image\_crops**]**  # Pack image\_crops so that first dimension is for position-senstive boxes.  position\_sensitive\_features **=** tf**.**stack**(**image\_crops**,** axis**=**0**)**  # Unroll the position-sensitive boxes to spatial positions.  position\_sensitive\_features **=** tf**.**squeeze**(**  tf**.**batch\_to\_space\_nd**(**position\_sensitive\_features**,**  block\_shape**=[**1**]** **+** num\_spatial\_bins**,**  crops**=**tf**.**zeros**((**3**,** 2**),** dtype**=**tf**.**int32**)),**  squeeze\_dims**=[**0**])**  # Reorder back the depth channel.  **if** block\_size **>=** 2**:**  position\_sensitive\_features **=** tf**.**depth\_to\_space**(**  position\_sensitive\_features**,** block\_size**=**block\_size**)**  **return** **position\_sensitive\_features** |

Rfcn的将[image\_height, image\_width, depth]的feature maps,但实际上是24, 19x19, 576,

首先, 把预测的box区域切出3x3共9个小bins.

然后, 把maps 在chn上分出9个小tensors.

接着, bins和tensors做crop, 生成对应尺寸的crop的更小的张量.

然后, 把这些更小张量求一个均值(维度上一一对应的位置相加,然后除以更小张量个数).

#### tf.image.crop\_and\_resize

|  |
| --- |
| **tf.image.crop\_and\_resize**介绍  该函数从输入图像里裁剪出一部分区域然后再重新缩放并返回处理后的图像。  形参：  (image, boxes, box\_ind, crop\_size, method="bilinear", extrapolation\_value=0, name=None)  image：shape为**[batch, image\_height, image\_width, depth]**且其dtype只能为`uint8`, `int8`, `int16`, `int32`, `int64`, `half`, `float32`, `float64`其一；  **boxes**：一系列标注框其shape为[num\_boxes, 4]，每个标注框的数据对应坐标为[y1, x1, y2, x2]，函数将以标注框来裁剪部分区域；**## 这里是[1083 4]**  box\_ind：指定引用标注框里的哪一个坐标系，其shape为[num\_boxes]；  crop\_size：裁剪区域要缩放的大小，形如size = [crop\_height, crop\_width]，所有裁剪区域均被缩放到此大小；  method：图像缩放所采用的插值方法，目前仅支持bilinear；  extrapolation\_value：预留；  return：dtype为float32且其shape为[num\_boxes, crop\_height, crop\_width, depth]的4-D张量，得到的是一系列定义区域统一缩放后的一批图像； |

#### tf.reduce\_mean

|  |
| --- |
| tensorflow笔记 ：reduce\_mean()函数axis参数理解  tf.reduce\_mean(input\_tensor, axis=None, keepdims=False, name=None, reduction\_indices=None)  作用：沿着张量不同的数轴进行计算平均值。  **它有两个含义, 目的是reduce, 方法是求均值,即通过求mean的方法做reduce.**  看到不少答案  感觉这个参数axis意义的解释不太清楚，只是说明了结果的规律  总结了一下，希望可以表达清楚  （1）  axis缺省值为none，表示对所有元素求平均  （2）  axis=0，表示对第一维度（行）减少，减少行的方法是对所有列求平均，即在行上压缩减少为一行。  1 2 3 在行上压缩减少为一行  4 5 6 ↑  箭头表示数据求平均的方向  （3）  若axis=1，表示对第二维度（列）减少，减少列的方法是对所有行求平均，在列上压缩减少为一列  1 2 3  4 5 6  在列上压缩减少为一列 ←  箭头表示数据求平均的方向 |

## 8.2 重点看position\_sens map生成

##### A) Map生成

在utils\ops.py中.

|  |
| --- |
| total\_bins **=** 1  bin\_crop\_size **=** **[]**  **for** **(**num\_bins**,** crop\_dim**)** **in** zip**(**num\_spatial\_bins**,** crop\_size**):**  **if** num\_bins **<** 1**:**  **raise** ValueError**(**'num\_spatial\_bins should be >= 1'**)**  **if** crop\_dim **%** num\_bins **!=** 0**:**  **raise** ValueError**(**'crop\_size should be divisible by num\_spatial\_bins'**)**  total\_bins **\*=** num\_bins  bin\_crop\_size**.**append**(**crop\_dim **//** num\_bins**)**  **if** **not** global\_pool **and** bin\_crop\_size**[**0**]** **!=** bin\_crop\_size**[**1**]:**  **raise** ValueError**(**'Only support square bin crop size for now.'**)**  ymin**,** xmin**,** ymax**,** xmax **=** tf**.**unstack**(**boxes**,** axis**=**1**)## boxes本身是[num\_boxes,4]的格式.分解成(unstack to)四个表示尺寸的[num\_boxes]格式.**  spatial\_bins\_y**,** spatial\_bins\_x **=** num\_spatial\_bins  # Split each box into spatial\_bins\_y \* spatial\_bins\_x bins.  position\_sensitive\_boxes **=** **[]**  **for** bin\_y **in** range**(**spatial\_bins\_y**):**  step\_y **=** **(**ymax **-** ymin**)** **/** spatial\_bins\_y  **for** bin\_x **in** range**(**spatial\_bins\_x**):**  step\_x **=** **(**xmax **-** xmin**)** **/** spatial\_bins\_x  box\_coordinates **=** **[**ymin **+** bin\_y **\*** step\_y**,**  xmin **+** bin\_x **\*** step\_x**,**  ymin **+** **(**bin\_y **+** 1**)** **\*** step\_y**,**  xmin **+** **(**bin\_x **+** 1**)** **\*** step\_x**,**  **]**  position\_sensitive\_boxes**.**append**(**tf**.**stack**(**box\_coordinates**,** axis**=**1**))## 四个表示尺寸的[num\_boxes]格式的组合成(stack to)[num\_boxes,4],然后再叠加成(append to) [9,num\_boxes,4]的格式.**  **Image\_splits是reg [9 19 19 80] cls [9 19 19 21]**  image\_splits **=** tf**.**split**(**value**=image,** **# reg [19 19 720], cls[19 19 189]**  num\_or\_size\_splits**=**total\_bins**,** axis**=**2**)**    image\_crops **=** **[]**  **for** **(**split**,** box**)** **in** zip**(**image\_splits**,** position\_sensitive\_boxes**):## reg时pos box格式是[9 1083 4] ,这里先分解成(for zip语句) [1083,4]的格式(box的shape).另外image\_splits是[9 19 19 80],分解成(for zip语句) [19 19 80]**  **if** split**.**shape**.**is\_fully\_defined**()** **and** box**.**shape**.**is\_fully\_defined**():**  crop **=** matmul\_crop\_and\_resize**(**  tf**.**expand\_dims**(**split**,** 0**),** box**,** bin\_crop\_size**)**  **else:**  crop **=** tf**.**image**.**crop\_and\_resize**(**  tf**.**expand\_dims**(**split**,** 0**),** box**,**  tf**.**zeros**(**tf**.**shape**(**boxes**)[**0**],** dtype**=**tf**.**int32**),** bin\_crop\_size**)**  **## split[19 19 80],box[1083 4], split先在dim0扩出来, split[1 19 19 80]**   1. **为什么要扩维度?**   **(*因为 tf.image.crop\_and\_resize函数需要的是 [batch, img\_w,img\_h,chn]的格式,我们原始是[19 19 80],不满足的,因此扩充(tf.expand\_dims)成四个维度*)**   1. **返回值是**   ***[num\_boxes, crop\_height, crop\_width, depth]***  ***因此也就解释了为什么一个crop是[1083 6 6 80]的格式.***   1. **bin\_crop\_size是6.经计算得到的.**   image\_crops**.**append**(**crop**) ## 一个crop是 reg[1083 6 6 80] ,cls [1083 6 6 21]**  **## image\_crops格式是 reg[9 1083 6 6 80], [9 1083 6 6 21],九个crops.**  **if** global\_pool**:**  # Average over all bins.  position\_sensitive\_features **=** tf**.**add\_n**(**image\_crops**)** **/** len**(**image\_crops**) ## 长度是9**  pos1 **=** position\_sensitive\_features **## 此时的pos,reg[1083 6 6 80], cls [1083 6 6 21]维度竟然少了一维(因tf.add\_n作用),都叠加在dim0上了.就少了dim0**  # Then average over spatial positions within the bins.  position\_sensitive\_features **=** tf**.**reduce\_mean**(**  position\_sensitive\_features**,** **[**1**,** 2**],** keep\_dims**=True)**  **## 最终的pos reg[1083 1 1 80], cls [1083 1 1 21]**  **在h,w维度(dim1,dim2)上做压缩,把本来的[num\_boxes, crop\_h, crop\_w,depth]在h,w领域压缩, 参考之前的例子. 在某个域上压缩,并不会计算该域上的”和”以及均值,而是计算其他域对应位置的均值.**  **具体操作可以这样理解:**   1. **最后一个depth是80个成员是为一组,dim2表示crop\_w为6,说明有6组(每组是80个). 把这6组元素对应位置求均值.之后.crop\_w就还有一组了(成员是均值的80个元素)** 2. **Crop\_h还有6组,每组是一组crop\_w,每个crop\_w是80个均值.让这6组再求和进而求均值.这样crop\_h就还有一组了.** 3. **这样就达到了[1083 1 1 80]的效果.**   **进一步可想到,这两个1(croph,cropw)已经没意义了,sequeeze它们就好了.**  tfprint**.**ssd\_debug0 **=** tf**.**Print**(**image**,[**"image,imagesplit"**,**tf**.**shape**(**image**),**tf**.**shape**(**image\_splits**),**tf**.**shape**(**crop**),**tf**.**shape**(**image\_crops**),**len**(**image\_crops**),**tf**.**shape**(**pos1**),**tf**.**shape**(**position\_sensitive\_features**)],**summarize**=**8**)** |

##### 实验数据

对于回归问题:

|  |
| --- |
| [image,imagesplit][19 19 720][9 19 19 80][1083 6 6 80][9 1083 6 6 80][9][1083 6 6 80][1083 1 1 80]  再添加一个position\_sensitive\_boxes的shape  [image,imagesplit][19 19 720][9 19 19 80][1083 6 6 80][9 1083 6 6 80][9][1083 6 6 80][1083 1 1 80]**[9 1083 4]** |

对于分类问题:

|  |
| --- |
| [image,imagesplit][19 19 189][9 19 19 21][1083 6 6 21][9 1083 6 6 21][9][1083 6 6 21][1083 1 1 21] |