**SSD使用说明**

# 一、Caffe环境搭建

**SSD源码：**<https://github.com/weiliu89/caffe/tree/ssd>

## 1.步骤

### 1）创建容器

**cpu：**

docker run -it -v /home:/opt --name caffe\_cpu\_zj --restart=always -e PASSWORD=a1b2c3 caffe\_ssd

**gpu:**

nvidia-docker run -it -v /home:/opt --name caffe\_cpu\_zj --restart=always -e PASSWORD=a1b2c3 caffe\_ssd

### 2）安装

**获得代码：**

git clone https://github.com/weiliu89/caffe

cd caffe

git checkout ssd

**编译：**

cp Makefile.config.example Makefile.config

make -j8

make py

make test -j8

make runtest -j8

**备注**：make pycaffe （编译python版本的caffe）

### 3）修改环境变量:

vim /etc/profile

export PYTHONPATH= /opt/zj/caffe/caffe /python:$PYTHONPATH

source /etc/profile

echo $PYTHONPATH （显示）

### 4）如果修改源码等，需重新编译caffe:

make clean

make all -j8

make test -j8

make runtest

## 2.遇到的问题

如：1. Check failed: error == cudaSuccess (35 vs. 0) CUDA driver version is insufficient for CUDA runtime version

**原因：**cuda没安装成功

**解决办法：**Makefile.config中CPU\_ONLY := 1 放开

2.“error == cudaSuccess (2 vs. 0) out of memory”

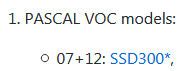
**解决办法：**显存不足的错误，把train\_val.prototxt中的batch\_size从256一路改到4才成功

3. 在训练模型的时候，无故就奔溃了：

**解决办法：**是因为被训练的图片大小是：136\*38，尺寸太小。我最后把它扩展成256\*256，模型训练成功。

## 3.测试

先下载SSD300\*300的模型：



**测试：**

python examples/ssd/ssd\_pascal.py

python examples/ssd/score\_ssd\_pascal.py（演示detection的训练结果，数值在0.718左右）

python examples/ssd/ssd\_pascal\_webcam.py （演示网络摄像头识别效果）

**error1:** Check failed: error == cudaSuccess (10 vs. 0)

**解决办法：**

打开caffe/examples/ssd/ssd\_pascal.py这个文件，找到gpus=’0,1,2,3’这一行，如果您的服务器有一块显卡，则将123删去，如果有两个显卡，则删去23，以此类推。如果您服务器没有gpu支持，则注销以下几行，程序会以cpu形式训练。（这个是解决问题cudasuccess（10vs0）的方法）

#Ifnum\_gpus >0:

# batch\_size\_per\_device =int(math.ceil(float(batch\_size) / num\_gpus))

#iter\_size =int(math.ceil(float(accum\_batch\_size) / (batch\_size\_per\_device \* num\_gpus)))

# solver\_mode =P.Solver.GPU

# device\_id =int(gpulist[0])  
保存后终端运行：python examples/ssd/ssd\_pascal.py

如果出现问题cudasuccess（2vs0）则说明您的显卡计算量有限，再次打开caffe/examples/ssd/ssd\_pascal.py这个文件，找到batch\_size =32这一行，修改数字32，可以修改为16，或者8，甚至为4，保存后终端运行：python examples/ssd/ssd\_pascal.py

**输入图像大小512\*512，batchsize=4，占内存为6G。**

**error2:** Check failed: mdb\_status == 0 (2 vs. 0) No such file or directory

**原因：** 没有VOC数据

**解决办法：**

python examples/ssd/ssd\_detect.py ：gpu版本->cpu版本

**需要修改的地方如下：**

1. 屏掉2行代码：

#caffe.set\_device(gpu\_id)

#caffe.set\_mode\_gpu()

1. parser.add\_argument('--gpu\_id', type=int, default=-1, help='gpu id') （默认的0改成-1）

**修改检测结果保存路径：**img.save('examples/detect\_result.jpg')

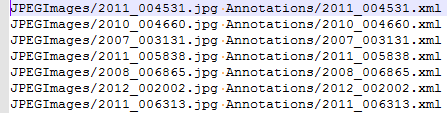
# 二、准备数据

### 准备数据

**文件路径：**/data/yushan/caffe/data/actions/preprocess\_data\_zj.py

将图片放到JPEGImages文件夹、将xml放到Annotations文件夹，自己写代码生成train.txt、val.txt.

(在总数据集中按比例分成训练集、验证机) \*.txt 内容如下：

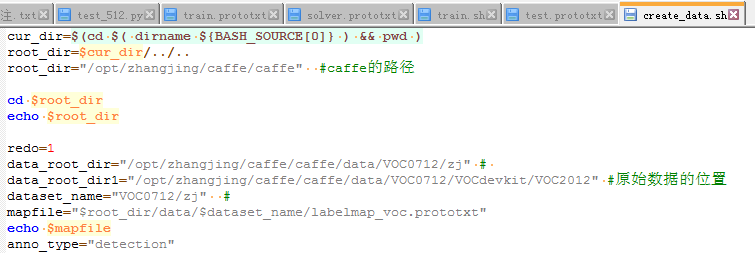


### 生成lmdb

**文件路径：**/data/yushan/**caffe/data/actions/create\_data.sh**

使用create\_data.sh生成lmdb可以跑SSD训练。不需要事先对图像归一化处理，代码会根据train.prototxt的图像宽高来归一化的。

**修改地方如下：**



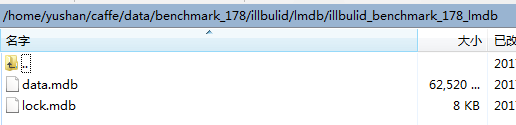


C:\Users\zhangjing1\AppData\Roaming\JunDaoIM\tempImages\image_lo.png

C:\Users\zhangjing1\AppData\Roaming\JunDaoIM\tempImages\image_Xw.png

**结果：**

生成的lmdb数据：

****

**下面是上面生成lmdb数据的快捷方式：**

**C:\Users\zhangjing1\AppData\Roaming\JunDaoIM\tempImages\image_OX.png**

**备注：**

**训练或测试时，数据路径写到illbuild\_benchmark\_178\_lmdb文件夹或者该文件夹对应的快捷方式。**

**如果出现下面错误：**

**File "/opt/zhangjing/ruike/caffe/scripts/create\_annoset.py", line 165, in <module>**

**process = subprocess.Popen(cmd.split(), stdout=subprocess.PIPE)**

**File "/usr/lib/python2.7/subprocess.py", line 710, in \_\_init\_\_**

**修改后代码：process = subprocess.Popen(cmd.split(), stdout=subprocess.PIPE，shell=True)**

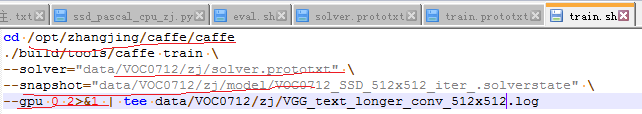
**出现问题：/opt/zhangjing/ruike/caffe/build/tools/convert\_annoset: not found**

**原因是：caffe/tools/convert\_annoset.cpp是ssd的文件，普通的caffe版本是没有该文件的。**

# 三、训练

## 1、训练脚本：train.sh

新建一个train.sh文件，按照下面的内容来写，运行train.sh就可以跑训练了。



-gpu 0,1

-gpu all

### 参数说明

**--solver:** 超参数的路径

**--snapshot:** 断点训练，会将模型的权重、训练参数都添加，保存模型的索引会在预训练

模型基础上增加；

**--weights:** 增量训练，只添加模型的权重，保存模型的索引从头开始。

例子：

--weights="jobs\_12\_512\_new\_lg/VGG\_VOC0712\_SSD\_512x512\_lg\_iter\_9700**0.caffemodel**"

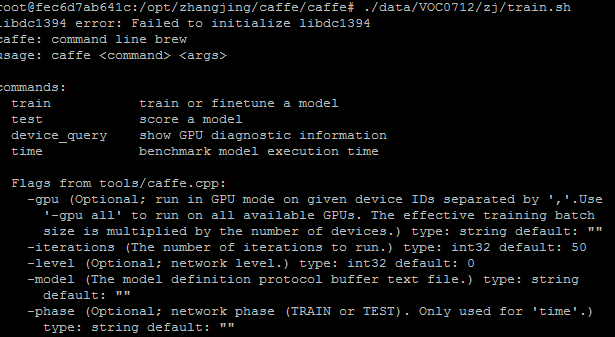
如果是重新训练，添加模型预训练模型：

VGG\_ILSVRC\_16\_layers\_fc\_reduced.caffemodel

**备注：-**-weights、--snapshot同时只能存在一个。

### error：

**错误如下图：**



**原因**是：.sh文件中—weights等号前后有空格。

**备注：**(命令行里面的写法）

./build/tools/caffe train --solver data/VOC0712/zj/solver.prototxt

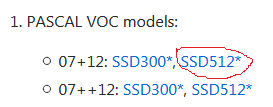
## 2、准备工作

### 下载VGGNet预训练模型

[链接：http://pan.baidu.com/s/1miDE9h2 密码：0hf2](https://gist.github.com/weiliu89/2ed6e13bfd5b57cf81d6+%20+%E2%80%9Cfully%20convolutional%20reduced%20%28atrous%29%20VGGNet%E2%80%9D)，将它放入caffe/models/VGGNet/目录下

### 在github上下载网络

以PASCAL VOC models中的07+12（SSD512）为例：

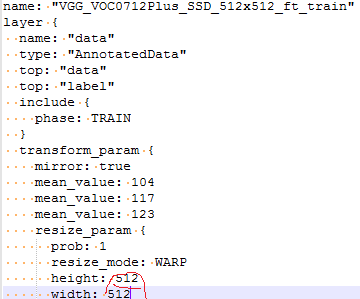


**备注：**

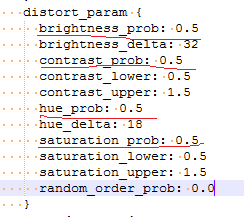
该模型是基于VGG16的，如果想使用googlenet等可以在model文件夹下面找相应的网络结构（train.prototxt）

## 3、修改训练网络：train.prototxt

### 修改1：输入图像大小



### 修改2：样本增强

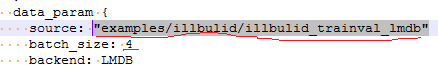


比如： hue\_prob：0.5 是按照0.5的概率随机生成

### 修改3：加载数据路径、标签索引文件路径

#### **修改加载数据路径**、格式、batch\_size

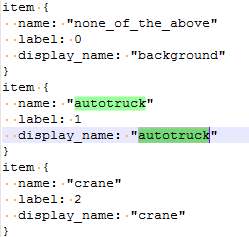
**注意：** 如果报找不到对应路径，就写全路径。 create\_data.sh生成lmdb数据及其的快捷方式，有红色的为快捷方式，下面路径为快捷方式所在路径，也可以写真实lmdb生成的位置（data、lock外面那个位置）。



#### 2) 修改标签索引文件路径

C:\Users\zhangjing1\AppData\Roaming\JunDaoIM\tempImages\image_0v.png

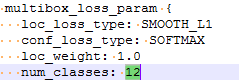
##### 类别名称：labelmap\_voc.prototxt文件



**备注：**类别名称为background、autotruck、crane、….

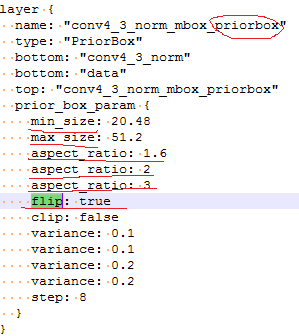
### 修改4: 类别个数

12 = 1（背景） + 11



### 修改5：anchor

例如：num\_classes: 12 (背景+目标)



#### 参数说明：

默认宽高比例为1:1

**max\_size**: 1:1

**注意**：1:1对应的实际宽高比为 :

**min\_size**: 1:1、1.6:1、2:1、3:1

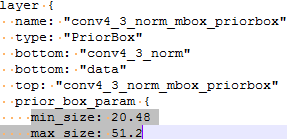
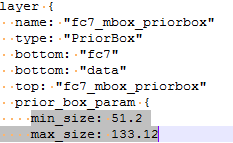
**注意**：2:1对应的实际宽高比为min\_size\* : min\_size/

**flip==true: （min\_size）** 1:1.6、1:2、1:3

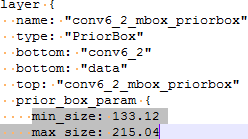
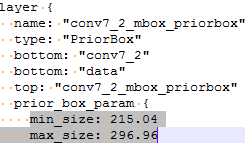
**共8个anchor**

**aspect\_ratio:** 宽高比，这个比例可以通过统计训练集中目标的宽高比（将原图大小缩放到网络输入大小）来调这个参数比较好。min\_size参数也可以用同样的方法来。

1. clip参数的作用目前还不是很清楚，一般也没修改过；
2. d = max\_size – min\_size (“conv4\_3\_normal\_mbox\_priorbox”的max\_size参数有时没有，所以该层的d与其它层的d不同，其它层的d一般保持恒定)，比如：

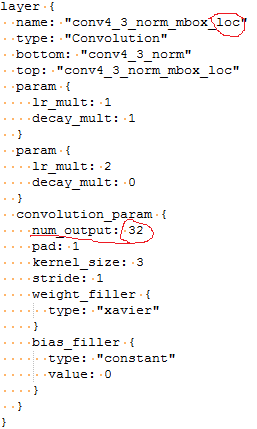
1. （2）

（3） （4）

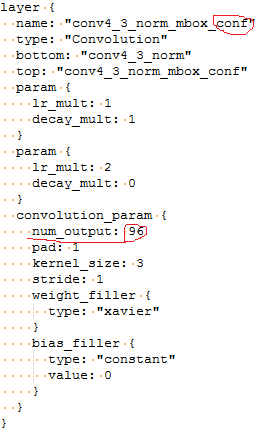
### 修改6：mbox\_loc

**num\_output: 32 (**4\*8, 4为(dx，dy,dw,dh）,8为anchor的个数**)**



### 修改7：mbox\_conf

**num\_output: 96** (12\*8, 12为类别，8为anchor的个数)



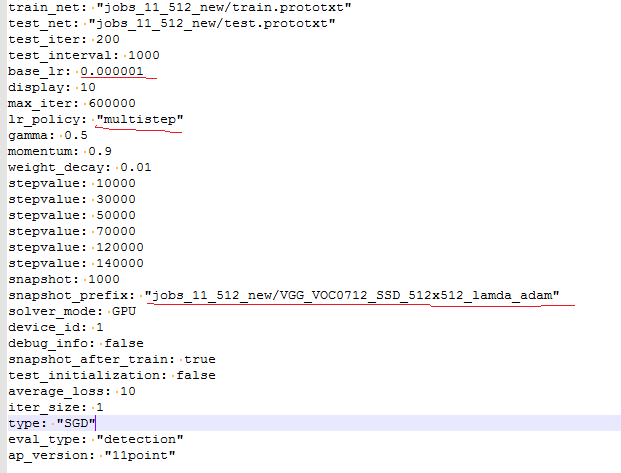
## 4、修改超参数：solver.prototxt

**备注：**

如果是测试mAP值，max\_iter：0； (不需要反向传播)

如果是训练，max\_iter: 600000。

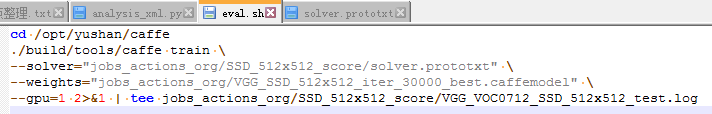
如果在训练时不使用test\_iter、test\_interval参数，训练过程就不需要测试。



# 四、测试（统计mAP）

## 1．测试脚本：eval.sh

新建一个eval.sh文件，按照下面的内容来写，运行eval.sh就可以跑测试了。



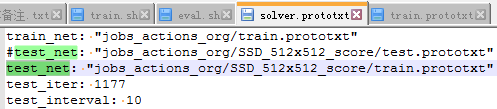
**备注：**需注意solver.prototxt的路径以及solver.prototxt文件中网络结构的路径。

### error

1. F0127 02:52:39.024227 16240 insert\_splits.cpp:29] Unknown bottom blob 'data' (layer 'conv1\_1', bottom index 0)

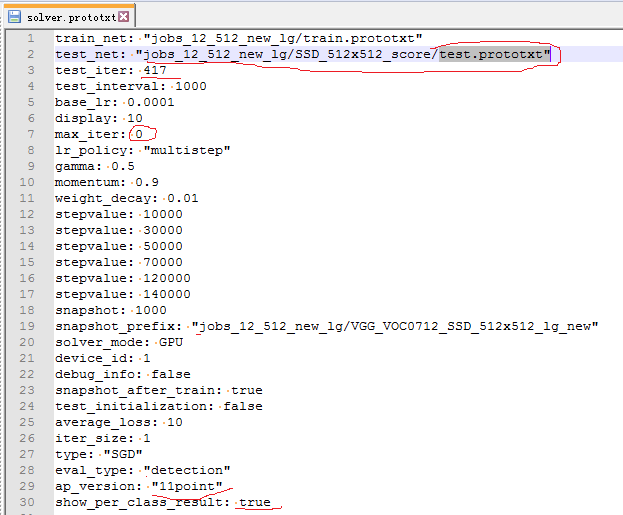
**原因：**层的名字对不上

**解决办法：**检查solver.prototxt文件中的路径以及对应的文件是否正确。

****

**备注：**1) solver.prototxt必须要有train\_net、test\_net，但是上面train\_net、test\_net对应的train.prototxt是不一样的，具体不一样地方请参考上面[修改test.prototxt](#_修改test.prototxt)。同时，**train\_net、test\_net加载的lmdb数据也不能一样**。

## 2．修改超参数：solver.prototxt



**备注：**

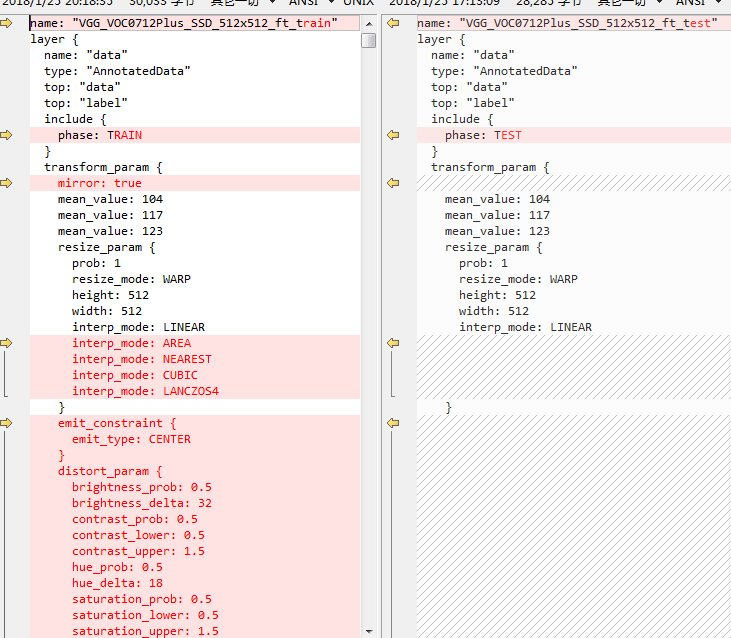
1. **test\_net：** 测试网络，注意路径
2. **test\_iter：** 测试样本的个数
3. max\_iter: 0 （必须为0， 因为是测试，不需要反向传播）
4. ap\_version：“11point” (计算mAP采用11point方法)
5. show\_per\_class\_result: true （统计并显示每个类别的mAP）

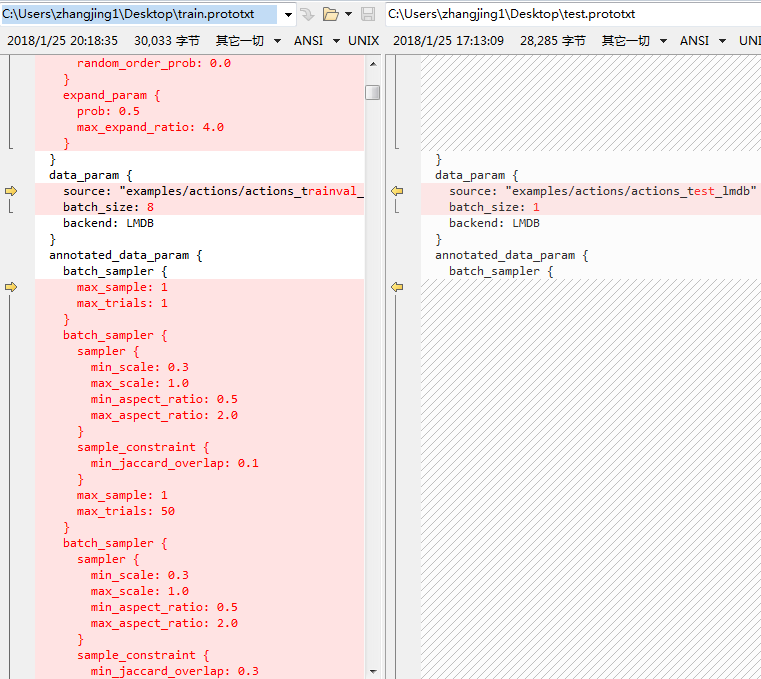
## 3．修改测试网络：test.prototxt

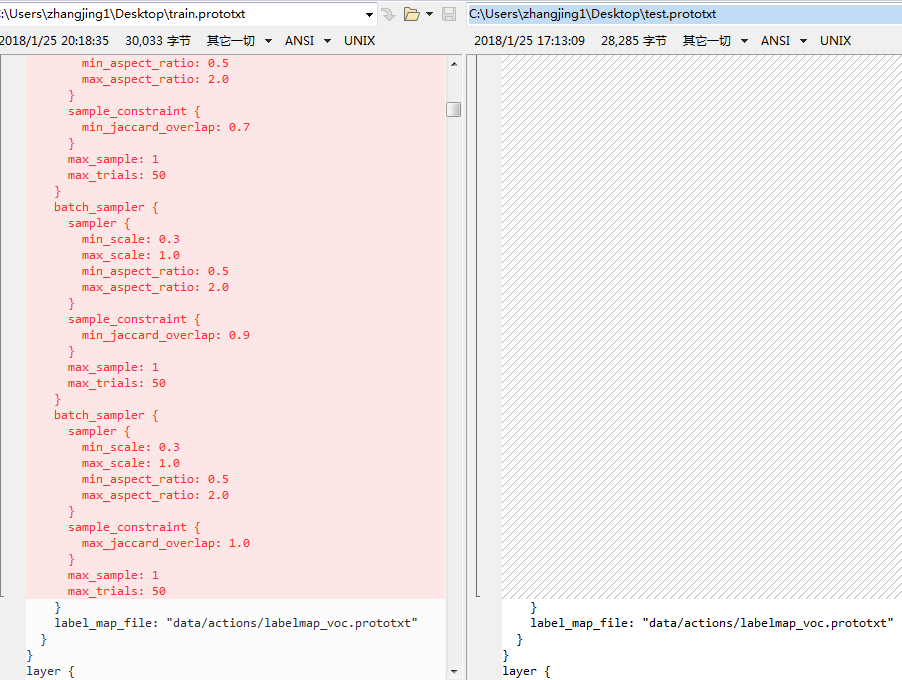
### 新建测试网络：test.prototxt

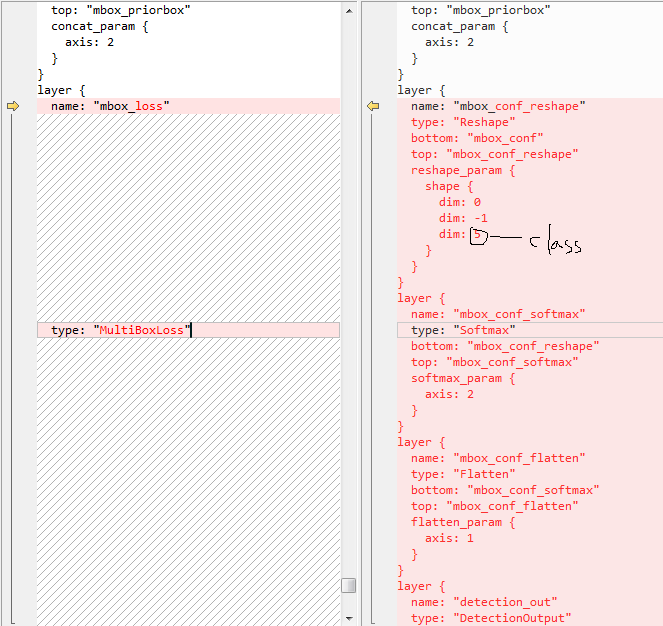
按照下面[train.prototxt与test.prototxt的不同](#_train.prototxt与test.prototxt对比结果如下：)，将train.prototxt改成test.prototxt。

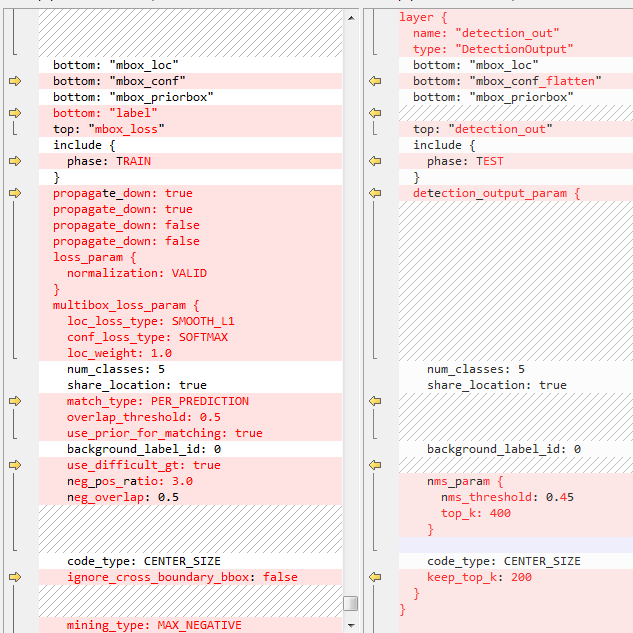
#### train.prototxt与test.prototxt对比结果如下：

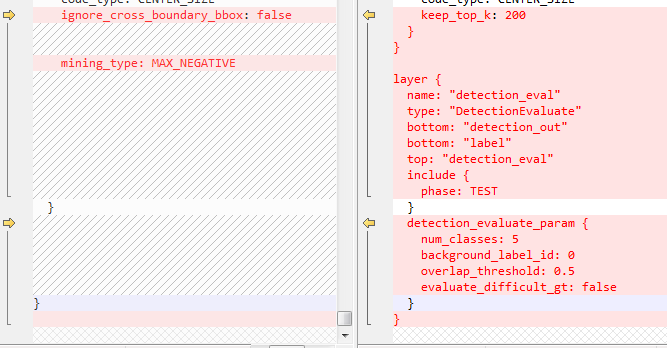












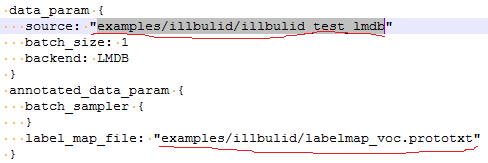
**备注：** 注意修改测试数据的路径，别的修改的地方在训练时都已经修改。

1)、训练时，需要对图像进行增强处理，目的是增加样本；测试时，不需要做这些处理。

2)、训练时，需要计算box的loss（就会有"mbox\_loss"层）；测试时，需要有"detection\_out"层。

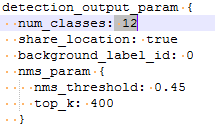
### 修改1：加载数据路径、标签索引文件路径

**备注：**对于测试模型，batch\_size=1

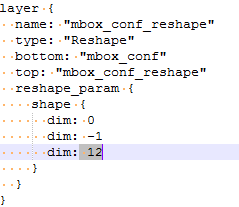


### 修改2：类别个数

**dim：** **12** （12为类别数目）



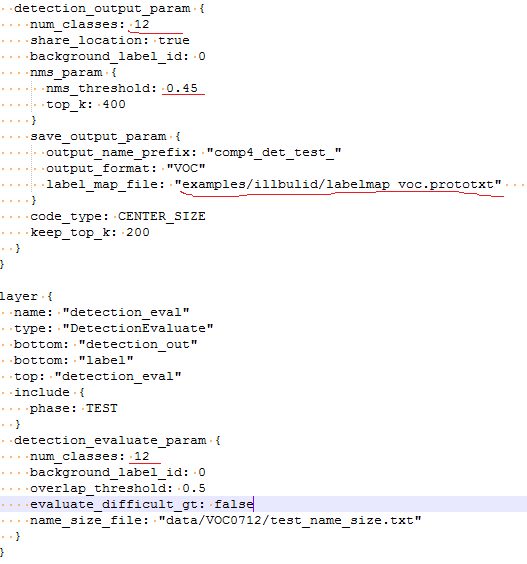
### 修改3：reshape维度



### 修改4：mbox\_loc、mbox\_conf、anchor

**参考训练网络train.prototxt**

### 其它



**注意：**test\_name\_size.txt文件一般没有，只要有就行，它是通过create\_list.sh文件生成的。

#### 备注：

/data/VOC0712文件夹下有create\_list.sh和create\_data.sh两个脚本，用于生产训练数据的list和实际数据的LMDB格式。

1. 手动生成list文件，共3个：”test.txt”, “test\_name\_size.txt”, “trainval.txt”；

2. 利用create\_data.sh生成训练用的LMDB格式文件（该文件调用的是SSD\_ROOT/scripts/create\_annoset.py文件，需要修改下路径的参数。（会生成lmdb的数据以及快捷方式）

name\_size\_file: 该参数一般也不需要修改

## 4.修改测试网络：deploy.prototxt

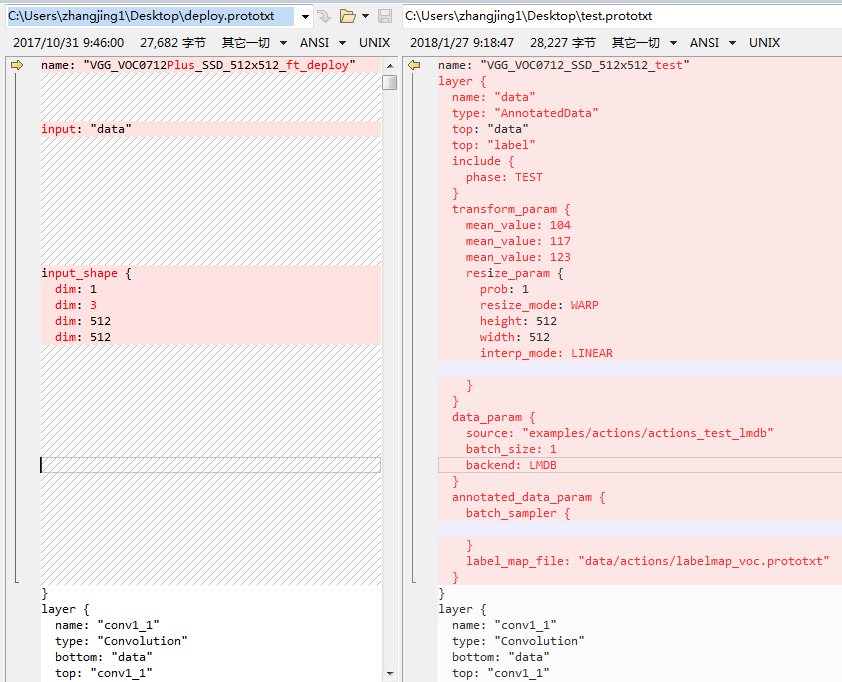
### 新建deploy.prototxt文件

根据下面test.prototxt与deploy.prototxt文件的区别，将test.prototxt修改成deploy.prototxt。

#### test.prototxt与deploy.prototxt对比

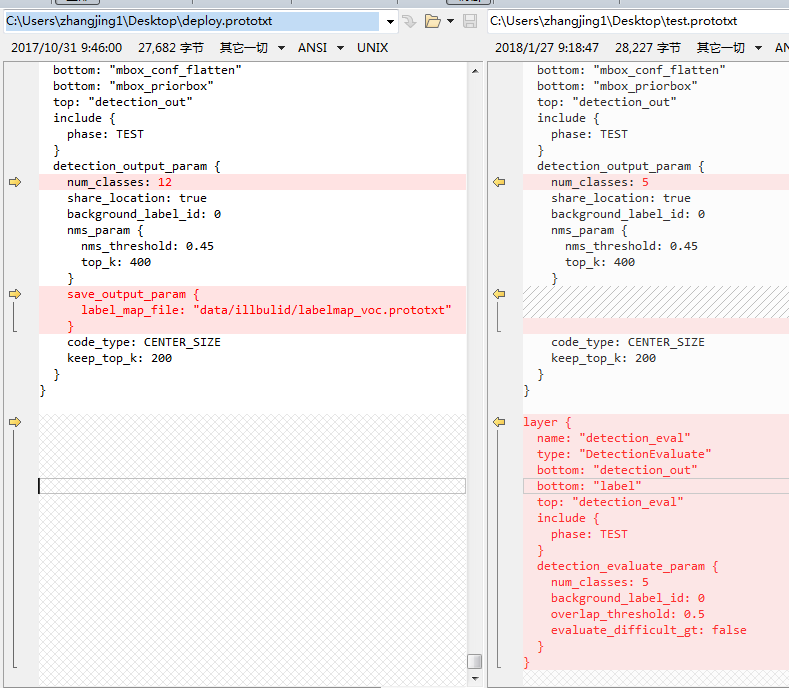
##### data层不同

**data层不同，deploy.prototxt与test.prototxt对比如下：**



##### 最后层不同

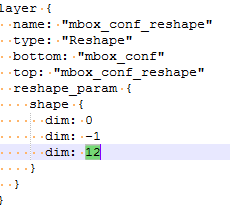
**最后层不同，deploy.prototxt与test.prototxt对比如下：**

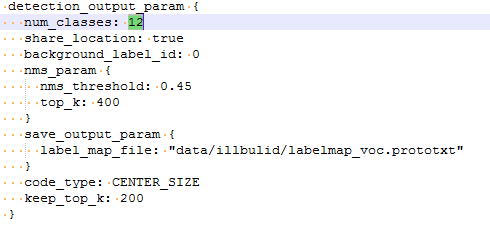


**备注：**save\_output\_param参数可以没有。

### 修改1：类别个数

**根据类别修改下面绿色的：**





### 修改2：其它

参考train.prototxt文件

## 5. train.prototxt、test.prototxt、deploy.prototxt 对比

train.prototxt、test.prototxt、deploy.prototxt 文件内容基本都一样，train.prototxt用于训练，test.prototxt用于测试，deploy.prototxt用于发布服务等。

train.prototxt : 训练，需要计算loss

test.prototxt: 测试， 需要计算准确率, 多了DetectionEvaluate层，少了一些训练用的；

deploy.prototxt: 与test.prototxt基本一样，不同在于数据层不一样，该数据层是输入的图

片，该网络结构用于测试。并且比test.prototxt少“detection\_eval”层。比

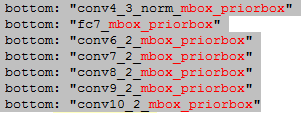
如在deepDetect部署时需要，不需要计算loss、准确率。

[在将检测结果画到图上，使用了deploy.prototxt。](#_test_512.py代码)

# 其它知识点

## SDD使用的VGG16

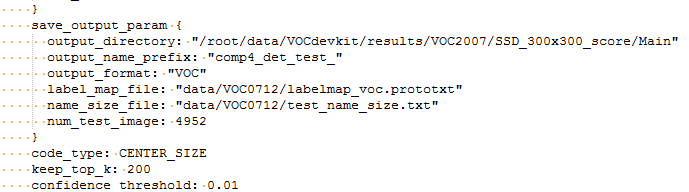
通过这7层来做预测：



该网络结构是在VGG16基础上取掉全连接层，而添加了几层卷积。 coco数据集目标小，最小设置20\*20，用了7层；VOC数据集目标大，最小设置30\*30，用了6层；前些层预测小目标，后些层预测大目标。

## confidence\_threshold:0.01

(deploy.prototxt、test.prototxt) 用于测试，在nms处理之前先根据confidence\_threshold过滤一些。



**备注：**

caffe/layers/detection\_output\_layer.hpp ： （confidence\_threshold:0.01）->ApplyNMSFast()->bbox\_util.cpp

1) 保留置信度大于confidence\_threshold的box等；

2) top\_k：如果第一步剩余的个数大于top\_k，再只保留最大的top\_k个，否则全部保留。（升序）

3) 再进行nms处理。

## mAP值参数：ap\_version

ap\_version的参数：默认参数是Integral

**参考文献：** https://sanchom.wordpress.com/tag/average-precision/

mAP = 前后召回率的间隔\*准确率; #其实就是离散函数积分求面积

**11point：**只用到准确率，11-point interpolated average precision，默认召回率的间隔是{0, 0.1, 0.2, …, 0.9, 1.0}，VOC2007 style for computing AP.

**MaxIntegral：**使用准确率、召回率；VOC2012 or ILSVRC style for computing AP. 从后往前计算。

**Integral：**与MaxIntegral差不多，只是从前往后计算；

## solver.prototxt 参数解释

**train\_net/test\_net/net**：train.prototxt #网络的配置文件

**type**: SGD #优化方法，如adam

**test\_iter**：100 #每次测试batch\_size\*test\_iter个样本

**test\_interval: 1000** #测试网络，也就是说每训练1000次，才进行一次测试

**备注：**

如果在训练时不使用test\_iter、test\_interval参数，训练过程就不需要测试，测试是不会调节loss的，只是验证模型。

**base\_lr: 0.01** #设置基础学习率, 如果训练一开始就跑飞，就将该值一直降低，不如0

到0.0001，一般添加预训练模型该值都会很低。

**lr\_policy: “inv”**  #学习率调整的策略：

**-fixed:** 保持base\_lr不变

**-step:** 如果设置step,则需要设置一个stepsize,返回

base\_lr\*gamma^((floor(iter/stepsize)), 其中iter是当前迭代次数；

**-exp:** 返回base\_lr\*gamma^ite**r**

**-inv：** 如果设置为inv,还需要设置一个power,返回base\_lr\*（1+gamma\*iter）

^(-power)

**-multistep：**如果设置为multistep,则还需要设置一个stepvalue，这个参数和step

很相似，step是均等间隔变化,而multistep是根据stepvalue值变化。

**-poly:** 学习率进行多项式误差，返回base\_lr（1-iter/max\_iter）^power

**-sigmoid:** 学习率进行sigmoid衰减，返回base\_lr

（1/1+exp(-gamma\*(iter-stepsize))）

**momentum: 0.9 #**动量

**display: 100 #**每训练100次，在屏幕上显示一次，如果设置为0，则不显示。

**max\_iter: 2000 #**最大迭代次数,2万次后就停止了，**注意**了在测试时需要将该参数设为0。

**snapshot: 1000** #快照，将训练出来的model和solver状态进行保存，用于设置训练多

少次后进行保存。

**snapshot\_prefix:** “examples/mnist/lennet” #存放保存模型的位置

**solver\_model:** CPU #设置运行模式，默认为GPU,如果你没有GPU,则需要改成CPU,否

则会出错。

**weight\_decay: 0.0005** #权值衰减，防止过拟合，正则化系数，权值衰减惩罚项使得权

值收敛到较小的绝对值，而惩罚大的权值。因为大的权值会使

得系统出现过拟合，降低其泛化性能。该值越大惩罚力度越大。

**clip\_gradients**: 35 # [default = -1]，梯度裁剪，若该参数大于零，把梯度限制在-

clip\_gradients到clip\_gradients之间。

**eval\_type:** "detection" #

**ap\_version:** "11point #采用“11point”方式计算mAP值

**show\_per\_class\_result**:true #显示每个类别的mAP值

## layer

### 5.1 convolution layer

layer {

name: "conv1"

type: "Convolution" #卷积

bottom: "data" #输入

top: "conv1" #输出

param { lr\_mult: 1 }#学习率的系数，最终的学习率是

lr\_mult\*base\_lr(slover.prototxt中的)。如果有两

个lr\_mult，第一个表示权值的学习率系数，第二个表示偏

值的。如果lr\_mult为0，在训练过程中就不会调该层的权

重。

param { lr\_mult: 2}

convolution\_param {

num\_output: 96 # 卷积核的个数

kernel\_size: 11 # 卷积核的大小，11\*11

stride: 4 # 卷积核的步长，默认为1.

pad: 0 # 扩充边缘，默认为0，不扩充。

weight\_filler {

type: "gaussian" # 权值初始化，默认为“constant”，值全为0.

std: 0.01 # distribution with stdev 0.01 (default mean: 0)

}

bias\_filler {

type: "constant" # 偏值初始化，一般使用"constant"，值全为0.

value: 0

}

}

}

输入：n\*c0\*w0\*h0

输出：n\*c1\*w1\*h1

其中，c1就是参数中的num\_output,生成的特征图个数。

w1 = (w0+2\*pad-kernel\_size)/stride + 1

h1 = (h0+2\*pad-kernel\_size)/stride + 1

### reshape layer

layer {

    name: "reshape"

    type: "Reshape"

    bottom: "data"

    top: "output"

    reshape\_param {

        shape {

            dim: -1  (由系统自动计算维度，数据的总量不变)

            dim: 0 （保持维度不变）

            dim: 1

            dim: 1

        }

    }

}

数据维度：n\*c\*w\*h

假设原数据为：32\*3\*28\*28，表示32张3通道的28\*28的彩色图。

shape{

dim:0

dim:0

dim:14

dim:-1

}

输出数据为：32\*3\*14\*56

### 5.3 data layer

layer {

  name: "demo"

  type: "ImageData"

  top: "data"

  top: "label"

  include {

    phase: TRAIN

  }

#数据预处理，注意：如果训练采用将数据转到0~1，减均值等处理，预测时也要做相应的处理。

  transform\_param {

    scale: 0.00390625  #1.0/255, 将数据转换到0~1之间。

    mean\_value: 128  #均值，彩色图会有3个均值，或者使用mean\_file

mirror:1 #1表示开启镜像，0表示关闭镜像，也可以使用true/false，（水平镜像，

\*do\_mirror = param\_.mirror() && Rand(2); 按照0.5的概率翻转，

data\_transformer.cpp中）

crop:227 #裁剪一个227\*227的图像块，在训练阶段随机裁剪，在测试

阶段从中间裁剪。

  }

  image\_data\_param {

source: "data/age\_to\_gender\_train.txt" #数据库来源

    new\_height: 48  #如果设置，则将图片进行resize

    new\_width: 48

    is\_color: false

    batch\_size: 256  #每次批量处理的个数

shuffle: true  #随机打乱顺序，默认值为false （image\_data\_param中采用shuffle参数，在

layers/image\_data\_layer.cpp中）

**备注：**create\_data.sh生成lmdb时有shuffle参数.

  }

或data\_param {

source: "examples/actions\_new/actions\_new\_train1\_lmdb"

batch\_size: 4

backend: LMDB

}

distort\_param { #数据增强

brightness\_prob: 0.5

brightness\_delta: 32

contrast\_prob: 0.5

contrast\_lower: 0.5

contrast\_upper: 1.5

hue\_prob: 0.5

hue\_delta: 18

saturation\_prob: 0.5

saturation\_lower: 0.5

saturation\_upper: 1.5

random\_order\_prob: 0.0

}

}

**备注：** caffe参数说明：<http://blog.csdn.net/qq_14845119/article/details/54929389>

**data:数据层**

type:数据属性，包含3种格式，

1种是Data，输入格式为LMDB，LEVELDB，

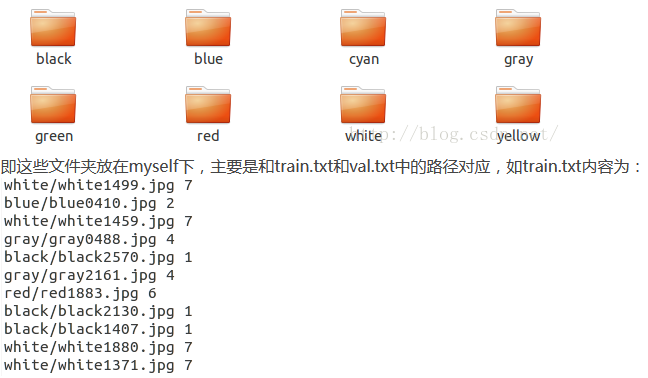
2种是ImageData，输入格式为\*.txt，txt中格式为，路径图片 类别标签，

3种为HDF5Data，该格式支持频谱图、特征向量的输入.

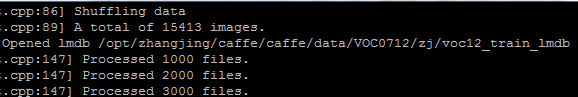
## 6. ****create\_imagenet.sh生成lmdb****

### 数据准备

**将examples/imagenet文件夹的create\_imagenet.sh 复制到自己的文件夹下，可以重命名为create\_lmdb.sh，并修改路径，开始生成lmdb. 处理的数据应该以下图的方式来放。（一般用于分类）下面的文件夹名就是一类，txt中标签数字代码类别号。**

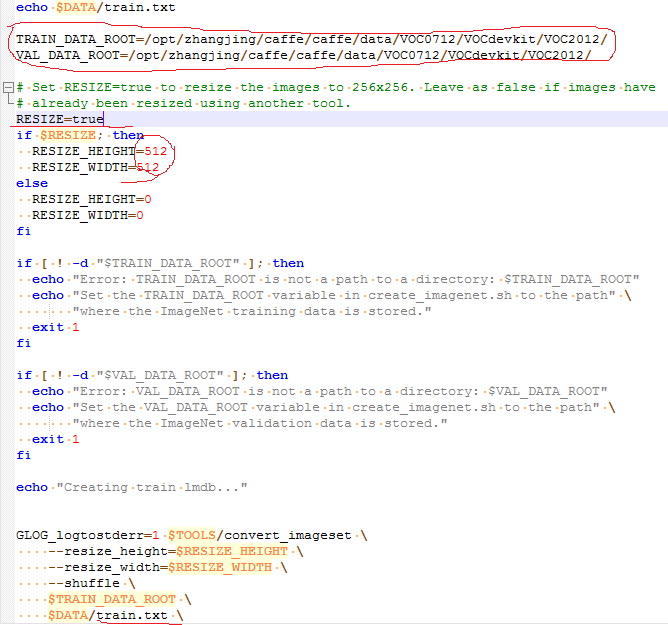
****

**结果如下：**



### ****修改地方如下：****

**1）修改路径 2）设置RESIZE=true，根据train.prototxt中输入图像的大小来修改RESIZE\_HEIGHT、RESIZE\_WIDTH.**

****

## 7.绘制网络结构图

### 搭建环境

**安装graphviz：**

**sudo apt-get install graphviz**

**安装pydot:**

**先安装pip**

**sudo apt-get install python-pip python-dev build-essential**

sudo pip install --upgrade pip

sudo pip install --upgrade virtualenv

再安装pydot

sudo pip install pydot

python的pydot模块是Graphviz的dot语言的python接口，主要使用Graphviz绘图，当然要安装pydot

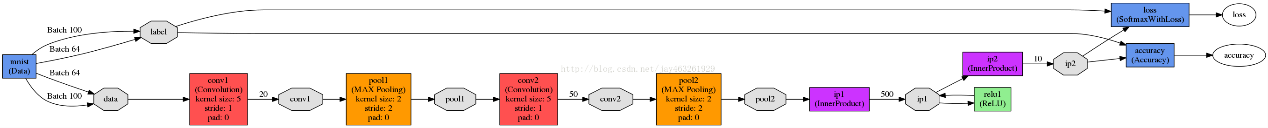
### 代码

python ./python/draw\_net.py  ./models/VGGNet/VOC0712/SSD\_300x300/train.prototxt   ssd\_net.png --rankdir=LR

**备注：**LR 从左到右

如果图片是bmp格式会出错：   return get\_pydot\_graph(caffe\_net, rankdir, phase=phase).create(format=ext)  
  File "/usr/local/lib/python2.7/dist-packages/pydot.py", line 1900, in create  
    assert p.returncode == 0, p.returncode  
AssertionError: 1

**结果：**



## 8. 在屏幕上显示程序运行时间

**time**参数用来在屏幕上显示程序运行时间。如：

# ./build/tools/caffe time -model examples/mnist/lenet\_train\_test.prototxt -iterations 10

这个例子用来在屏幕上显示lenet模型迭代10次所使用的时间。包括每次迭代的forward和backward所用的时间，也包括每层forward和backward所用的平均时间。

# ./build/tools/caffe time -model examples/mnist/lenet\_train\_test.prototxt -gpu 0

这个例子用来在屏幕上显示lenet模型用gpu迭代50次所使用的时间。

# ./build/tools/caffe time -model examples/mnist/lenet\_train\_test.prototxt -weights examples/mnist/lenet\_iter\_10000.caffemodel -gpu 0 -iterations 10

利用给定的权重，利用第一块gpu，迭代10次lenet模型所用的时间。

## 9.写python代码生成train.prototxt

**代码：**

# -\*- coding: UTF-8 -\*-

import caffe #导入caffe包

#创建网络结构函数

def **create\_net**(lmdb, mean\_file, batch\_size, include\_acc=False):

#网络规范

net = caffe.NetSpec()

#第一层：Data层,ntop=2（2个输出）

net.data, net.label = caffe.layers.Data(source=lmdb,

backend=caffe.params.Data.LMDB,

batch\_size=batch\_size, ntop=2,

transform\_param = dict(crop\_size = 40,

mean\_file=mean\_file, mirror=True))

#第二层：Convolution视觉层

net.conv1 = caffe.layers.Convolution(net.data, num\_output=20,

kernel\_size=5,weight\_filler={"type": "xavier"},

bias\_filler={"type": "constant"})

#第三层ReLU激活层

net.relu1 = caffe.layers.ReLU(net.conv1, in\_place=True)

#第四层Pooling池化层

net.pool1 = caffe.layers.Pooling(net.relu1, pool=caffe.params.Pooling.MAX,

kernel\_size=3, stride=2)

net.conv2 = caffe.layers.Convolution(net.pool1, kernel\_size=3,

stride=1,num\_output=32,

pad=1,weight\_filler=dict(type='xavier'))

net.relu2 = caffe.layers.ReLU(net.conv2, in\_place=True)

net.pool2 = caffe.layers.Pooling(net.relu2, pool=caffe.params.Pooling.MAX,

kernel\_size=3, stride=2)

#全连层

net.fc3 = caffe.layers.InnerProduct(net.pool2,

num\_output=1024,weight\_filler=dict(type='xavier'))

net.relu3 = caffe.layers.ReLU(net.fc3, in\_place=True)

#创建一个dropout层

net.drop3 = caffe.layers.Dropout(net.relu3, in\_place=True)

net.fc4 = caffe.layers.InnerProduct(net.drop3,

num\_output=10,weight\_filler=dict(type='xavier'))

#创建一个softmax层

net.loss = caffe.layers.SoftmaxWithLoss(net.fc4, net.label)

#训练的prototxt文件不包括Accuracy层,测试的时候需要。

if include\_acc:

net.acc = caffe.layers.Accuracy(net.fc4, net.label)

return str(net.to\_proto())

return str(net.to\_proto())

def **write\_net**(): #写prototxt文件

caffe\_root="/home /caffe-master/my-caffe-project/"#my-caffe-project目录

train\_lmdb = caffe\_root + "train.lmdb" #train.lmdb文件的位置

mean\_file = caffe\_root + "mean.binaryproto" #均值文件的位置

train\_proto = caffe\_root + "train.prototxt" #保存train\_prototxt文件的位置

test\_proto = caffe\_root + "test.prototxt" #保存test\_prototxt文件的位置

#写入prototxt文件

with open(train\_proto, 'w') as f:

f.write(str(create\_net(train\_lmdb, mean\_file, batch\_size=64)))

#写入prototxt文件

with open(test\_proto, 'w') as f:

f.write(str(create\_net(test\_proto, mean\_file, batch\_size=32, include\_acc=True)))

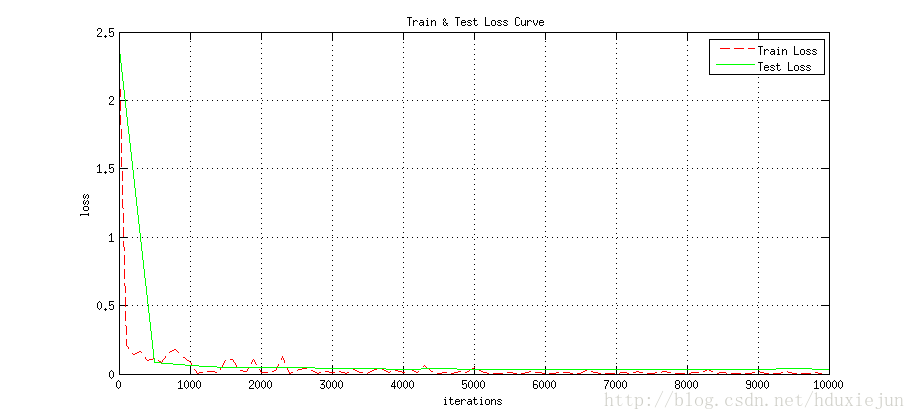
if \_\_name\_\_ == '\_\_main\_\_':

write\_net()

## 10.绘制loss、accuracy曲线

写python文件，根据训练生成的log文件绘制loss、accuracy曲线。

**学习曲线如下所示：**



**曲线解读：**

通过学习曲线，可以评估当前的训练状态：   
train loss 不断下降，test loss 不断下降，说明网络仍然在学习。   
train loss 不断下降，test loss 趋于不变，说明网络过拟合。   
train loss 趋于不变，test loss区域不变，说明学习曲线遇到瓶颈，需减小学习速率或批量数据尺寸。   
train loss 趋于不变，test loss不断下降，说明数据集100%有问题。   
train loss 不断上升，test loss 不断上升（最终变为NaN）可能是网络结构设计不当，训练超参数设计不当，程序bug等某个问题引起的，需要进一步等位。

# 可能要做的实验

## 1.lr\_base: 基础学习率

改基础学习率lr\_base，一般调小，比如0.0001，如果一开始跑训练，就跑飞了，这会lr\_base要更小；

## 2.修改网络结构

比如VGG16换成resnet-50.

## 3.改变图像大小

因为SSD网络输入图像是个正方形的，如果原始图像不是正方形的，缩放后目标可能会变形，所以需要将原图抠成正方形的。

还可以修改SSD数据层的宽高。（比如：300\*300、512\*512）

## 4.修改anchor个数以及min\_scale、max\_scale

## 5.不调节某些卷积层的权重

卷积层参数lr\_mult设置为0，就不会调节该层的权重。

## 6.增大Weight\_decay

防止过拟合。

## 7.改优化器

比如：adam优化器。

# 代码备份

## test\_512.py代码

import numpy as np

import matplotlib

matplotlib.use('Agg')

import os

import xml.dom.minidom

import cv2

import time

# %matplotlib inline

# Make sure that caffe is on the python path:

caffe\_root = '/opt/yushan/caffe' # this file is expected to be in {caffe\_root}/examples

os.chdir(caffe\_root)

import sys

sys.path.insert(0, 'python') #导入python模块

import caffe

#caffe.set\_device(0) #设置GPU的id

caffe.set\_mode\_cpu()

from google.protobuf import text\_format

from caffe.proto import caffe\_pb2

model\_def = '/opt/yushan/caffe/jobs\_12\_512\_newbbox\_lg/**deploy.prototx**t' #网络结构

model\_weights =

'/opt/yushan/caffe/jobs\_12\_512\_newbbox\_lg/VGG\_VOC0712\_SSD\_512x512\_lg\_3753p\_iter\_52000.caffemodel' #模型

# load PASCAL VOC labels

labelmap\_file = '/opt/yushan/caffe/examples/illbulid/labelmap\_voc.prototxt' #类别索引

file = open(labelmap\_file, 'r')

labelmap = caffe\_pb2.LabelMap()

text\_format.Merge(str(file.read()), labelmap)

def get\_labelname(labelmap, labels):

num\_labels = len(labelmap.item)

labelnames = []

if type(labels) is not list:

labels = [labels]

for label in labels:

found = False

for i in xrange(0, num\_labels):

if label == labelmap.item[i].label:

found = True

labelnames.append(labelmap.item[i].display\_name)

break

assert found == True

return labelnames

scales=((512,512),)

# function pick = nms(boxes,threshold,type)

def nms(boxes, threshold):

if boxes.size==0:

return np.empty((0,3))

x1 = boxes[:,0]

y1 = boxes[:,1]

x2 = boxes[:,2]

y2 = boxes[:,3]

s = boxes[:,4]

area = (x2-x1+1) \* (y2-y1+1)

I = np.argsort(s)

pick = np.zeros\_like(s, dtype=np.int16)

counter = 0

while I.size>0:

i = I[-1]

pick[counter] = i

counter += 1

idx = I[0:-1]

xx1 = np.maximum(x1[i], x1[idx])

yy1 = np.maximum(y1[i], y1[idx])

xx2 = np.minimum(x2[i], x2[idx])

yy2 = np.minimum(y2[i], y2[idx])

w = np.maximum(0.0, xx2-xx1+1)

h = np.maximum(0.0, yy2-yy1+1)

inter = w \* h

o = inter / (area[i] + area[idx] - inter)

I = I[np.where(o<=threshold)]

pick = pick[0:counter]

return pick

#网络结构

net = caffe.Net(model\_def, # defines the structure of the model

model\_weights, # contains the trained weights

caffe.TEST) # use test mode (e.g., don't perform dropout)

# input preprocessing: 'data' is the name of the input blob == net.inputs[0]

print(net.blobs['data'].data.shape)

print(model\_weights)

save\_dir='/opt/ligang/deepdetect/ssd/result\_newbbox\_178\_52000/'

orig\_image\_dir='/opt/yushan/caffe/data/benchmark\_178/pictures'

from os import path

files = [x for x in os.listdir(orig\_image\_dir) if path.isfile(orig\_image\_dir+os.sep+x)]

for line in files:

recttotal = []

imageLabels=[]

start = time.clock()

image\_path=orig\_image\_dir+os.sep+line

save\_detection\_path=save\_dir+line[0:len(line)-4]+'.txt'

image=caffe.io.load\_image(image\_path) #加载图片数据

saveimge = cv2.imread(image\_path)

image\_height,image\_width,channels=image.shape

print(max(image\_height,image\_width))

print(image\_height,image\_width)

detection\_result=open(save\_detection\_path,'wt')

for scale in scales:

image\_resize\_height = scale[0]

image\_resize\_width = scale[1]

#transformer为数据预处理，

transformer =

caffe.io.Transformer({'data': (1,3,image\_resize\_height,image\_resize\_width)})

transformer.set\_transpose('data', (2, 0, 1)) #h\*w\*c->c\*h\*w

transformer.set\_mean('data', np.array([104,117,123])) # mean pixel,与训练保持一致

transformer.set\_raw\_scale('data', 255) # the reference model operates on images in

[0,255] range instead of [0,1],数据归一化到

0~255，与训练保持一致

transformer.set\_channel\_swap('data', (2,1,0)) # the reference model has channels in

BGR order instead of RGB，通道转换

net.blobs['data'].reshape(1,3,image\_resize\_height,image\_resize\_width)

transformed\_image = transformer.preprocess('data', image)

net.blobs['data'].data[...] = transformed\_image

# Forward pass.

detections = net.forward()['detection\_out']

# Parse the outputs.

det\_label = detections[0,0,:,1]

det\_conf = detections[0,0,:,2]

det\_xmin = detections[0,0,:,3]

det\_ymin = detections[0,0,:,4]

det\_xmax = detections[0,0,:,5]

det\_ymax = detections[0,0,:,6]

# Get detections with confidence higher than 0.1.

top\_indices = [i for i, conf in enumerate(det\_conf) if conf >= 0.3]

top\_conf = det\_conf[top\_indices]

top\_label\_indices = det\_label[top\_indices].tolist()

top\_labels = get\_labelname(labelmap, top\_label\_indices)

top\_xmin = det\_xmin[top\_indices]

top\_ymin = det\_ymin[top\_indices]

top\_xmax = det\_xmax[top\_indices]

top\_ymax = det\_ymax[top\_indices]

for i in xrange(top\_conf.shape[0]):

xmin = int(round(top\_xmin[i] \* image.shape[1]))

ymin = int(round(top\_ymin[i] \* image.shape[0]))

xmax = int(round(top\_xmax[i] \* image.shape[1]))

ymax = int(round(top\_ymax[i] \* image.shape[0]))

xmin = max(1, xmin)

ymin = max(1, ymin)

xmax = min(image.shape[1]-1, xmax)

ymax = min(image.shape[0]-1, ymax)

score = top\_conf[i]

label\_name = top\_labels[i]

label\_name\_index = top\_label\_indices[i]

result=str(label\_name)+' '+str(xmin)+' '+str(ymin)+' '+str(xmax)+' '+str(ymin)+' '+str(xmax)+' '+str(ymax)+' '+str(xmin)+' '+str(ymax)+' '+str(score)+'\n'

#detection\_result.write(result)

#print(result)

cv2.rectangle(saveimge,(int(xmin), int(ymin)),(int(xmax),int(ymax)),(0,255,0),2)

cv2.putText(saveimge, str(score)[:3], (int(xmin), int(ymin)), cv2.FONT\_HERSHEY\_SIMPLEX,1,(255,0,0))

cv2.putText(saveimge, str(label\_name), (int(xmin)+50, int(ymin)), cv2.FONT\_HERSHEY\_SIMPLEX,1,(255,0,255))

#result1=str(label\_name)+' '+str(score)+'\n'

label\_len=len(str(label\_name\_index))

result1=str(label\_name\_index)[:(label\_len-2)]+' '+str(score)+'\n'

detection\_result.write(result1)

print(result)

end = time.clock()

print("read: %f s" % (end - start))

detection\_result.close()

'''xmin = 530

ymin = 460

xmax = 1490

ymax = 960

score ='0.86'

label\_name = 'brickspile'

cv2.rectangle(saveimge,(int(xmin), int(ymin)),(int(xmax),int(ymax)),(0,255,0),2)

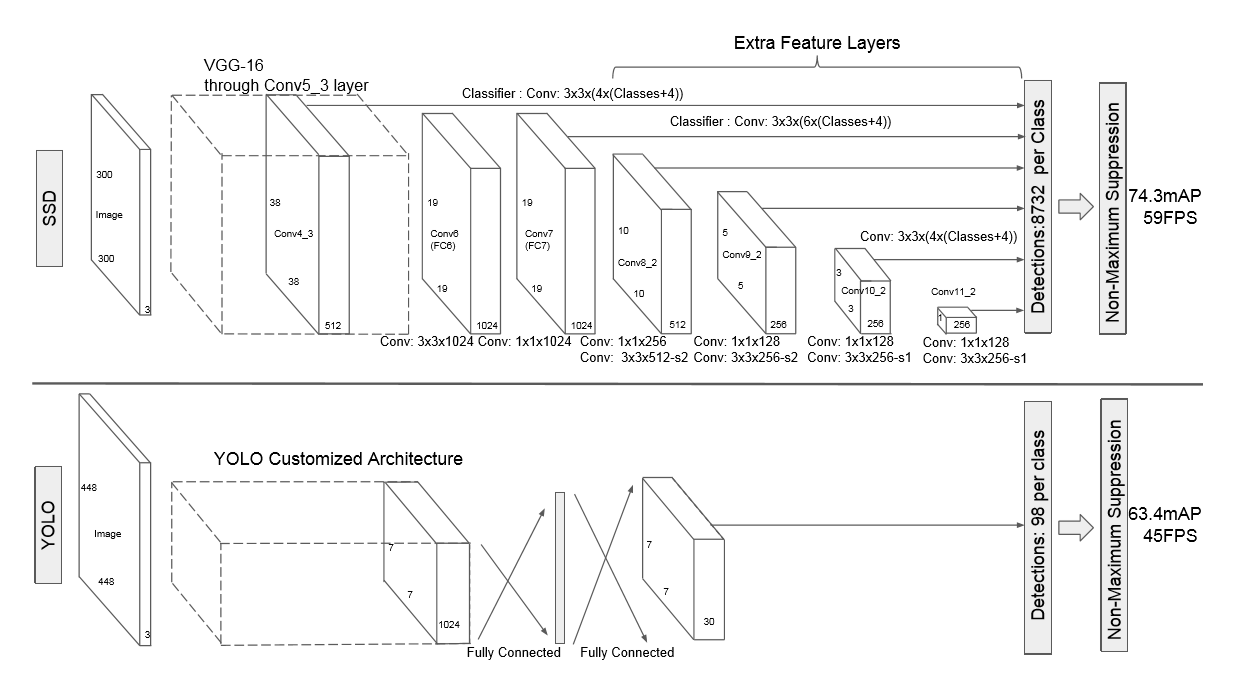
cv2.putText(saveimge, str(score)[:3], (int(xmin), int(ymin)), cv2.FONT\_HERSHEY\_SIMPLEX,1,(255,0,0))

cv2.putText(saveimge, str(label\_name), (int(xmin)+50, int(ymin)), cv2.FONT\_HERSHEY\_SIMPLEX,1,(255,0,255))'''

cv2.imwrite('/opt/ligang/deepdetect/ssd/result\_newbbox\_178\_52000/'+line,saveimge)

print('success')

# ssd与yolo结果图对比



# SSD标准网络与修改后网络对比

类别为4+1(背景)

loc = anchor\*4, 4为（dx,dy,dw,dh）

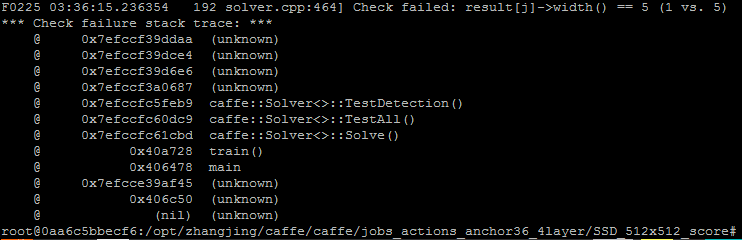
conf = anchor\*类别。

**以下类别为5类：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **标准的ssd，7个回归层** | | **修改后的，7个回归层** | |
| **网络** | **lr\_mult/ decay\_mult/lr\_mult / decay\_mult** | **convolution\_param**  **或**  **prior\_box\_param** | **lr\_mult/ decay\_mult/ lr\_mult/ decay\_mult** | **convolution\_param**  **或**  **prior\_box\_param** |
| **Conv1\_1+relu1\_1**  **Conv1\_2+relu1\_2**  **pool1** | (1,1,2,0)  (1,1,2,0) | 64,3\*3, p=1  64,3\*3,p=1  MAX, s=2 | **(0,0,0,0)**  **(0,0,0,0)** | 64,3\*3, p=1  64,3\*3,p=1  MAX, s=2 |
| **Conv2\_1+relu2\_1**  **Conv2\_2+relu2\_2**  **pool2** | (1,1,2,0)  (1,1,2,0) | 128,3\*3,p=1  128,3\*3,p=1  MAX, s=2 | **(0,0,0,0)**  **(0,0,0,0)** | 128,3\*3,p=1  128,3\*3,p=1  MAX, s=2 |
| **Conv3\_1+relu3\_1**  **Conv3\_2+relu3\_2**  **Conv3\_3+relu3\_3**  **pool3** | (1,1,2,0)  (1,1,2,0)  (1,1,2,0) | 256,3\*3,p=1  256,3\*3,p=1  256,3\*3, p=1  MAX, s=2 | **(0,0,0,0)**  **(0,0,0,0)**  **(0,0,0,0)** | 256,3\*3,p=1  256,3\*3,p=1  256,3\*3, p=1  MAX, s=2 |
| **Conv4\_1+relu4\_1**  **Conv4\_2+relu4\_2**  **Conv4\_3+relu4\_3**  **pool4** | (1,1,2,0)  (1,1,2,0)  (1,1,2,0) | 512, 3\*3, p=1  512, 3\*3, p=1  512, 3\*3, p=1  MAX, s=2 | (1,1,2,0)  (1,1,2,0)  (1,1,2,0) | 512, 3\*3, p=1  512, 3\*3, p=1  512, 3\*3, p=1  MAX, s=2 |
| **Conv5\_1+relu5\_1**  **Conv5\_2+relu5\_2**  **Conv5\_3+relu5\_3**  **pool5** | (1,1,2,0)  (1,1,2,0)  (1,1,2,0) | 512,3\*3,p=1  512,3\*3,p=1  512,3\*3,p=1  MAX, s=2 | (1,1,2,0)  (1,1,2,0)  (1,1,2,0) | 512, 3\*3, p=1  512, 3\*3, p=1  512, 3\*3, p=1  MAX, s=2 |
| **fc6+relu6** | (1,1,2,0) | 1024,3\*3,p=6 | (1,1,2,0) | 1024,3\*3,p=6 |
| **fc7+relu7** | (1,1,2,0) | 1024,1\*1 | (1,1,2,0) | 1024,1\*1 |
| **conv6\_1+conv6\_1\_relu**  **conv6\_2+conv6\_2\_relu** | (1,1,2,0)  (1,1,2,0) | 256, 1\*1, p=0, s=1  512, 3\*3, p=1, s=2 | (1,1,2,0)  (1,1,2,0) | 256, 1\*1, p=0, s=1  512, 3\*3, p=1, s=2 |
| **Conv7\_1+conv7\_1\_relu**  **Conv7\_2+conv7\_2\_relu** | (1,1,2,0)  (1,1,2,0) | 128, 1\*1, p=0, s=1  256, 3\*3, p=1, s=2 | (1,1,2,0)  (1,1,2,0) | 128, 1\*1, p=0, s=1  256, 3\*3, p=1, s=2 |
| **Conv8\_1+conv8\_1\_relu**  **Conv8\_2+conv8\_2\_relu** | (1,1,2,0)  (1,1,2,0) | 128, 1\*1, p=0, s=1  256, 3\*3, p=1, s=2 | (1,1,2,0)  (1,1,2,0) | 128, 1\*1, p=0, s=1  256, 3\*3, p=1, s=2 |
| **Conv9\_1+conv9\_1\_relu**  **Conv9\_2+conv9\_2\_relu** | (1,1,2,0)  (1,1,2,0) | 128, 1\*1, p=0, s=1  256, 3\*3, p=1, s=2 | (1,1,2,0)  (1,1,2,0) | 128, 1\*1, p=0, s=1  256, 3\*3, p=1, s=2 |
| **Conv10\_1+conv10\_1\_relu**  **Conv10\_2+conv10\_2\_relu** | (1,1,2,0)  (1,1,2,0) | 128, 1\*1, p=0, s=1  256, 4\*4, p=1, s=1 | (1,1,2,0)  (1,1,2,0) | 128, 1\*1, p=0, s=1  256, 4\*4, p=1, s=1 |
| **conv4\_3\_norm\_mbox\_loc**  **conv4\_3\_norm\_mbox\_conf** | (1,1,2,0)  (1,1,2,0) | 16, 3\*3, p=1, s=1  20, 3\*3, p=1, s=1 | (1,1,2,0)  (1,1,2,0) | 48, 3\*3,p=1, s=1  60, 3\*3,p=1, s=1 |
| **conv4\_3\_norm\_mbox\_priorbox** |  | min\_size: 35.84  max\_size: 76.8  aspect\_ratio: 2  flip: true  clip: false  step: 8  offset: 0.5  **（anchor=4个，5类）** |  | **min\_size: 20.48**  **max\_size: 51.2**  **aspect\_ratio: 1.6**  **aspect\_ratio: 2**  **aspect\_ratio: 3**  **aspect\_ratio: 4**  **aspect\_ratio: 5**  flip: true  clip: false  step: 8  **（anchor=12个，5类）** |
| **fc7\_mbox\_loc**  **fc7\_mbox\_conf** | (1,1,2,0)  (1,1,2,0) | 24,3\*3,p=1,s=1  30,3\*3,p=1,s=1 | (1,1,2,0)  (1,1,2,0) | 48,3\*3,p=1,s=1  60,3\*3,p=1,s=1 |
| **fc7\_mbox\_priorbox** |  | min\_size: 76.8  max\_size: 153.6  aspect\_ratio: 2  aspect\_ratio: 3  flip: true  clip: false  step: 16  offset: 0.5  **（anchor=6个，5类）** |  | **min\_size: 51.2**  **max\_size: 133.12**  **aspect\_ratio: 1.6**  **aspect\_ratio: 2**  **aspect\_ratio: 3**  **aspect\_ratio: 4**  **aspect\_ratio: 5**  flip: true  clip: false  step: 16  **（anchor=12个，5类）** |
| **conv6\_2\_mbox\_loc**  **conv6\_2\_mbox\_conf** | (1,1,2,0)  (1,1,2,0) | 24,3\*3,p=1,s=1  30,3\*3,p=1,s=1 | (1,1,2,0)  (1,1,2,0) | 48,3\*3,p=1,s=1  60,3\*3,p=1,s=1 |
| **conv6\_2\_mbox\_priorbox** |  | min\_size: 153.6  max\_size: 230.4  aspect\_ratio: 2  aspect\_ratio: 3  flip: true  clip: false  step: 32  offset: 0.5  **（anchor=6个，5类）** |  | **min\_size: 133.12**  **max\_size: 215.04**  **aspect\_ratio: 1.6**  **aspect\_ratio: 2**  **aspect\_ratio: 3**  **aspect\_ratio: 4**  **aspect\_ratio: 5**  flip: true  clip: false  step: 32  **（anchor=12个，5类）** |
| **conv7\_2\_mbox\_loc**  **conv7\_2\_mbox\_conf** | (1,1,2,0)  (1,1,2,0) | 24,3\*3,p=1,s=1  30,3\*3,p=1,s=1 | (1,1,2,0)  (1,1,2,0) | 24,3\*3,p=1,s=1  30,3\*3,p=1,s=1 |
| **conv7\_2\_mbox\_priorbox** |  | min\_size: 230.4  max\_size: 307.2  aspect\_ratio: 2  aspect\_ratio: 3  flip: true  clip: false  step: 64  offset: 0.5  **（anchor=6个，5类）** |  | **min\_size: 215.04**  **max\_size: 296.96**  **aspect\_ratio: 2**  **aspect\_ratio: 3**  flip: true  clip: false  step: 64  **（anchor=6个，5类）** |
| **conv8\_2\_mbox\_loc**  **conv8\_2\_mbox\_conf** | (1,1,2,0)  (1,1,2,0) | 24,3\*3,p=1,s=1  30,3\*3,p=1,s=1 | (1,1,2,0)  (1,1,2,0) | 16,3\*3,p=1,s=1  20,3\*3,p=1,s=1 |
| **conv8\_2\_mbox\_priorbox** |  | min\_size: 307.2  max\_size: 384.0  aspect\_ratio: 2  aspect\_ratio: 3  flip: true  clip: false  step: 128  offset: 0.5  **（anchor=6个，5类）** |  | **min\_size: 296.96**  **max\_size: 378.88**  **aspect\_ratio: 2**  flip: true  clip: false  step: 128  **（anchor=4个，5类）** |
| **conv9\_2\_mbox\_loc**  **conv9\_2\_mbox\_conf** | (1,1,2,0)  (1,1,2,0) | 16,3\*3,p=1,s=1  20,3\*3,p=1,s=1 | (1,1,2,0)  (1,1,2,0) | 8,3\*3,p=1,s=1  10,3\*3,p=1,s=1 |
| **conv9\_2\_mbox\_priorbox** |  | min\_size: 384.0  max\_size: 460.8  aspect\_ratio: 2  flip: true  clip: false  step: 256  offset: 0.5  **（anchor=4个，5类）** |  | **min\_size: 378.88**  **max\_size: 460.8**  flip: true  clip: false  step: 256  **（anchor=2个，5类）** |
| **conv10\_2\_mbox\_loc**  **conv10\_2\_mbox\_conf** | (1,1,2,0)  (1,1,2,0) | 16,3\*3,p=1,s=1  20,3\*3,p=1,s=1 | (1,1,2,0)  (1,1,2,0) | 8,3\*3,p=1,s=1  10,3\*3,p=1,s=1 |
| **conv10\_2\_mbox\_priorbox** |  | min\_size: 460.8  max\_size: 537.6  aspect\_ratio: 2  flip: true  clip: false  step: 512  offset: 0.5  **（anchor=4个，5类）** |  | **min\_size: 460.8**  **max\_size: 542.72**  flip: true  clip: false  step: 512  **（anchor=2个，5类）** |

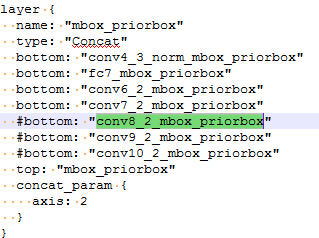
# Error

## 192 solver.cpp:464] Check failed: result[j]->width() == 5 (1 vs. 5)



**问题描述：**网络结构VGG16、7个分类回归层、去掉后面三个分类回归层，预测出现的情况。

**原因：**去掉后面三个分类回归层，只是将该三个分类回归层屏蔽了（如下图所示），多余代码还保留，删除多余代码该问题解决。

****

## ImportError:No module named \_caffe

**问题描述：**在代码中进行 import caffe 后报错ImportError:No module named \_caffe

**解决办法：重新编译**

cd caffe  #进入caffe目录，可能你的是cd caffe-master

    sudo make clean #清除原编译

    sudo make -j4  # make -j4或者make -j8 这根据你自己电脑来定

    sudo make pycaffe   #最关键的一个

    sudo  make runtest   #测试编译