**SSD使用说明**

# 一、Caffe环境搭建

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

**论文：**SSD single shot multibox detector

**论文链接：**<https://arxiv.org/abs/1512.02325>

## 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

### 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。

### 4. Check failed: status == CURAND\_STATUS\_SUCCESS (201 vs. 0) CURAND\_STATUS\_LAUNCH\_FAILURE

**原因：**应该是CUDA没有配置好。

**解决办法：**

sudo vim ~/.bashrc #打开配置文件，如果没安装vim，可执行

$ sudo apt-get install vim #安装vim

加入这两条语句：

export PATH=/usr/local/cuda-8.0/bin:$PATH

export LD\_LIBRARY\_PATH=/usr/local/cuda-8.0/lib64:$LD\_LIBRARY\_PATH

**让环境生效：**source ~/.bashrc

**查看CUDA版本**：$ nvcc --version

显示路径： $echo $PATH

### 5. Check failed: status == CURAND\_ST ATUS\_SUCCESS (102 vs. 0) CURAND\_STATUS\_ALLOCATION\_FAILED

**原因**：没内存

**解决办法**：停掉正在跑的程序，重新make runtest -j8.

### 6. root@95b2d7882483:/opt/zhangjing/caffe# sudo make runtest

**Error:**

.build\_release/tools/caffe

make: .build\_release/tools/caffe: Command not found

make: \*\*\* [runtest] Error 127

**解决办法：**重新编译

### 7. ImportError: No module named caffe.proto解决办法

**可能原因**：这种情况一般是没有把caffe中的和python相关的内容的路径添加到python

的编译路径中。

**解决办法：**

echo $ PYTHONPATH 查看python的路径；

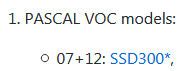
vim ~/.bashrc

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

source ~/.bashrc

## 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')

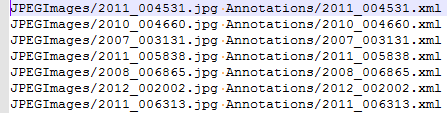
# 二、准备数据

### 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：

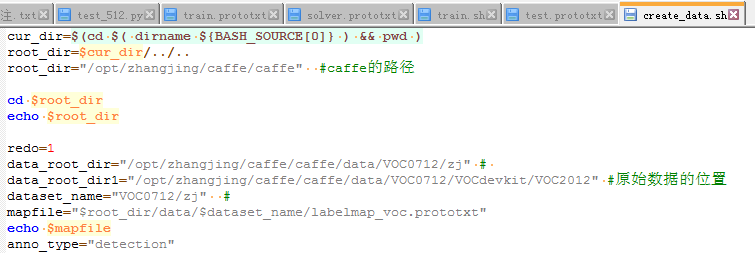


### 生成lmdb

**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：**



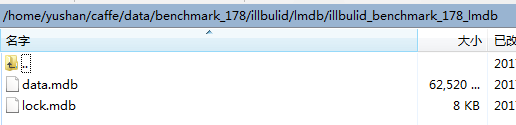


C:\Users\zhangjing1\AppData\Roaming\JunDaoIM\tempImages\image_lo.png

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)**

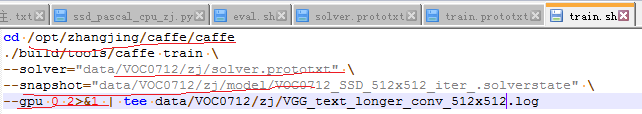
**出现问题：/opt/zhangjing/ruike/caffe/build/tools/convert\_annoset: not found**

**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**"

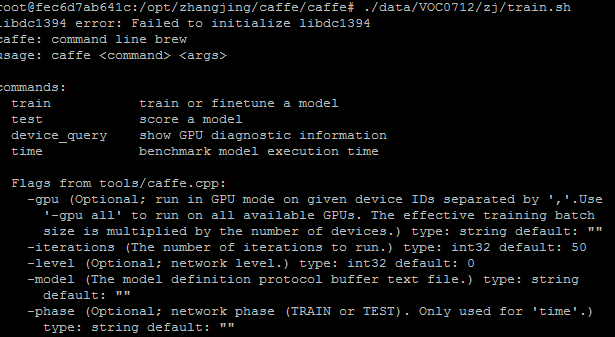
如果是重新训练，添加模型预训练模型：

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

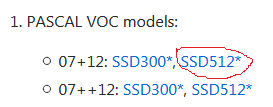
## 2、准备工作

### 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：

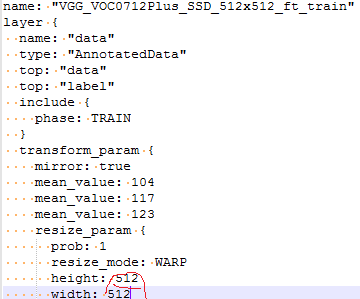


**备注：**

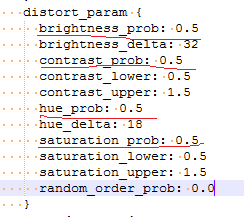
该模型是基于VGG16的，如果想使用googlenet等可以在model文件夹下面找相应的网络结构（train.prototxt）

## 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

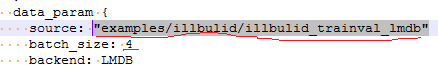


比如： hue\_prob：0.5 是按照0.5的概率随机生成

### 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

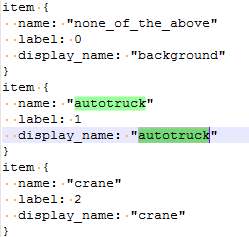
**注意：** 如果报找不到对应路径，就写全路径。 create\_data.sh生成lmdb数据及其的快捷方式，有红色的为快捷方式，下面路径为快捷方式所在路径，也可以写真实lmdb生成的位置（data、lock外面那个位置）。



#### 2) 修改标签索引文件路径

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

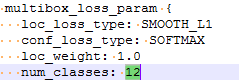
##### 类别名称：labelmap\_voc.prototxt文件



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

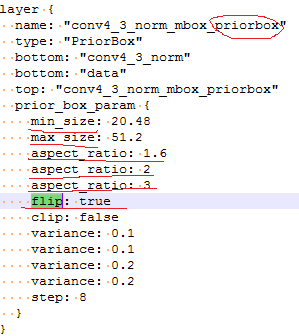
### 修改4: 类别个数

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

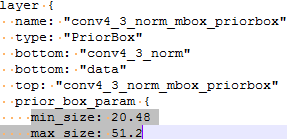
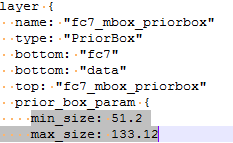
clip=false #做clip操作是为了让prior的候选坐标位置保持在[0,1]范围内。在caffe.proto文

件中有关于参数clip的解释，为”如果为true，则将先验框裁剪为[0，1]“。

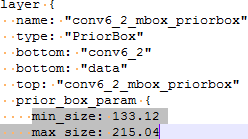
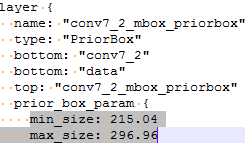
**共8个anchor**

**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：

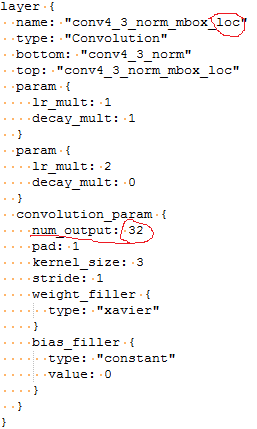
1. （2）

（3） （4）

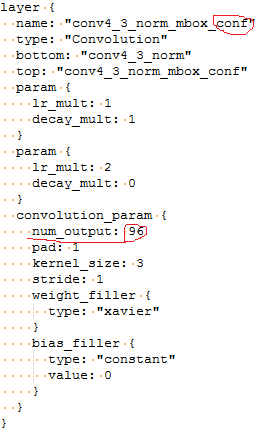
### 修改6：mbox\_loc

**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)



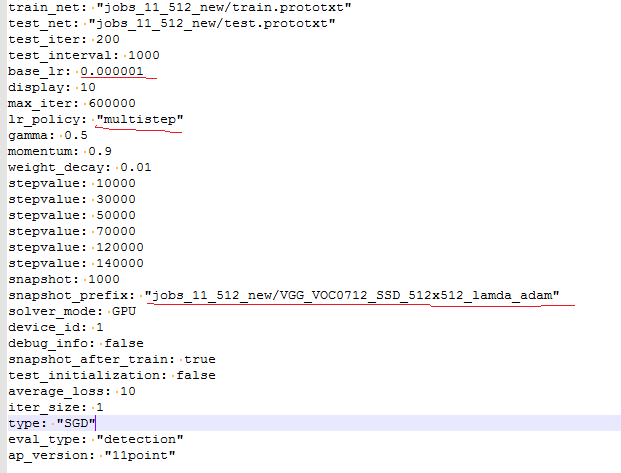
## 4、修改超参数：solver.prototxt

**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。

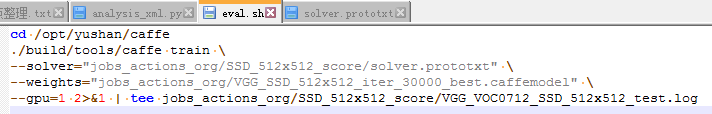
如果在训练时不使用test\_iter、test\_interval参数，训练过程就不需要测试。



# 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。



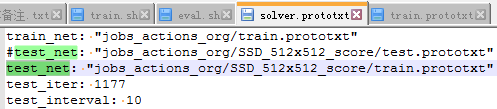
**备注：**需注意solver.prototxt的路径以及solver.prototxt文件中网络结构的路径。

### error

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

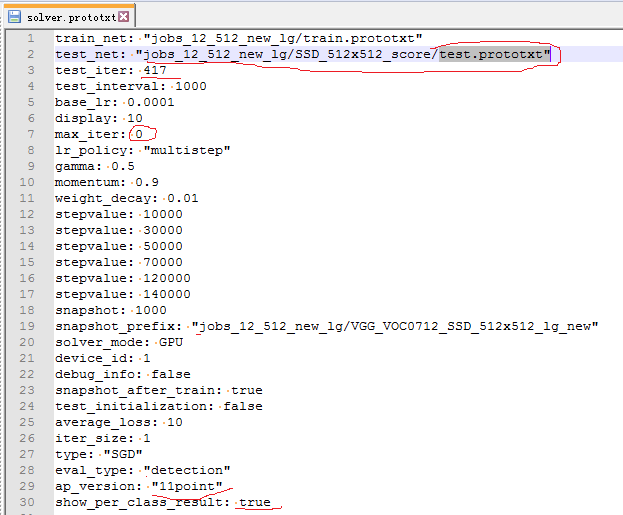
**原因：**层的名字对不上

**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：**

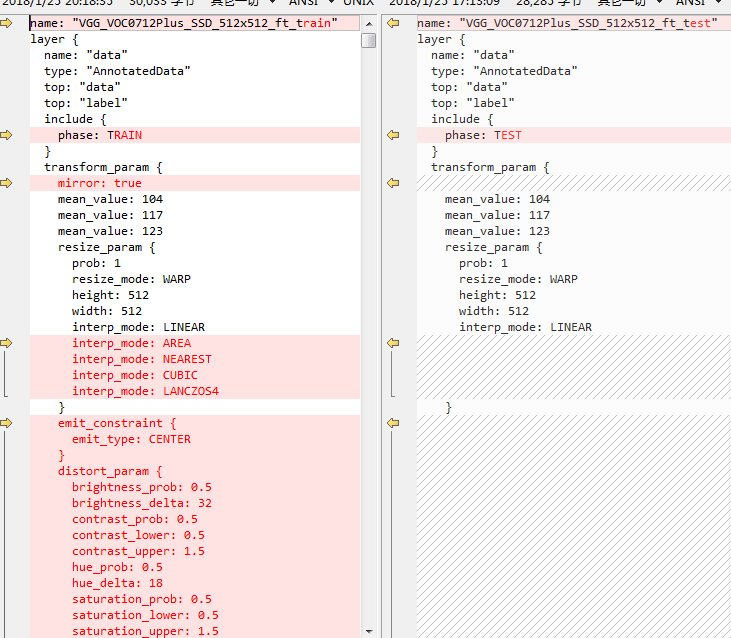
1. **test\_net：** 测试网络，注意路径
2. **test\_iter：** 测试样本的个数
3. max\_iter: 0 （必须为0， 因为是测试，不需要反向传播）
4. ap\_version：“11point” (计算mAP采用11point方法)
5. show\_per\_class\_result: true （统计并显示每个类别的mAP）

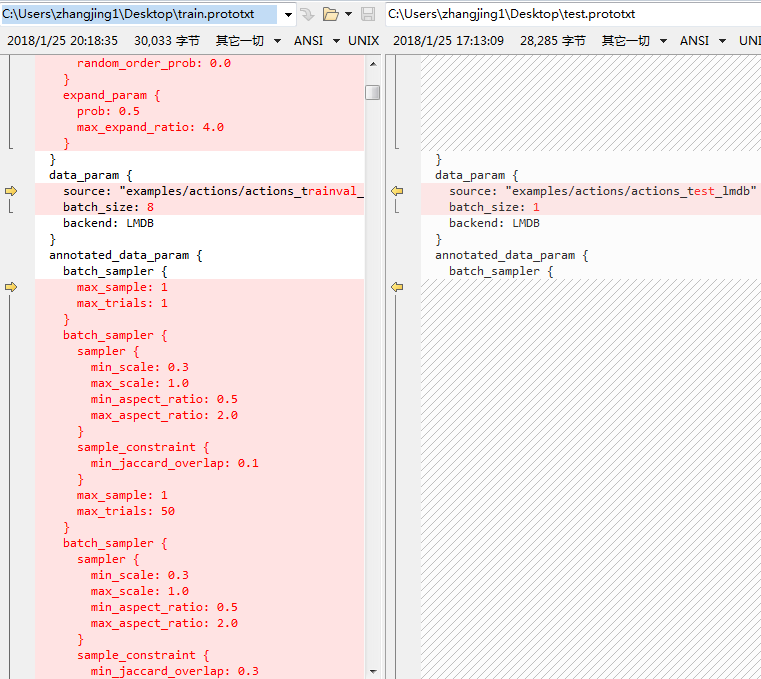
## 3．修改测试网络：test.prototxt

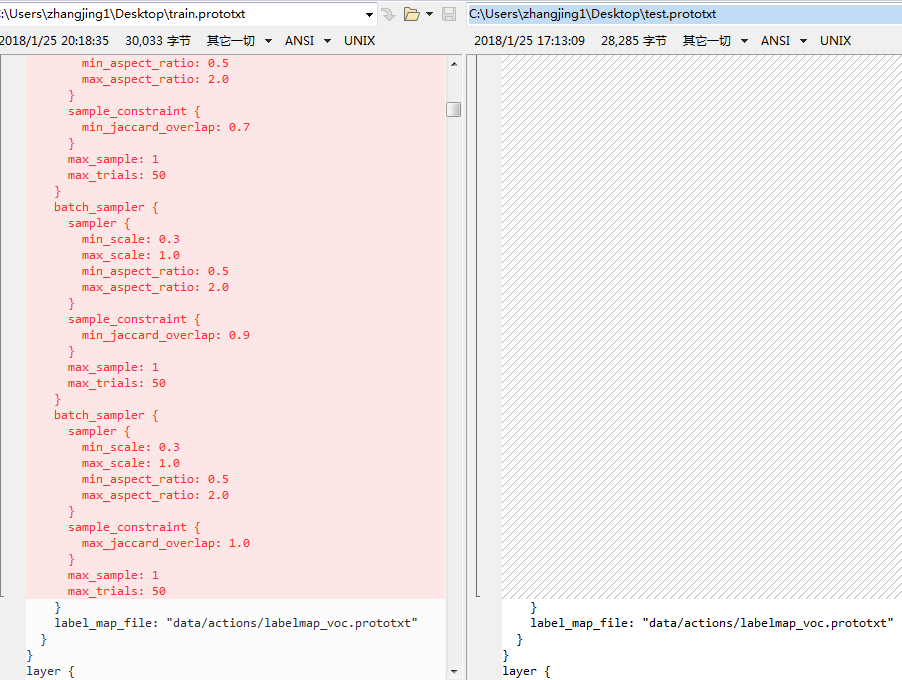
### 新建测试网络：test.prototxt

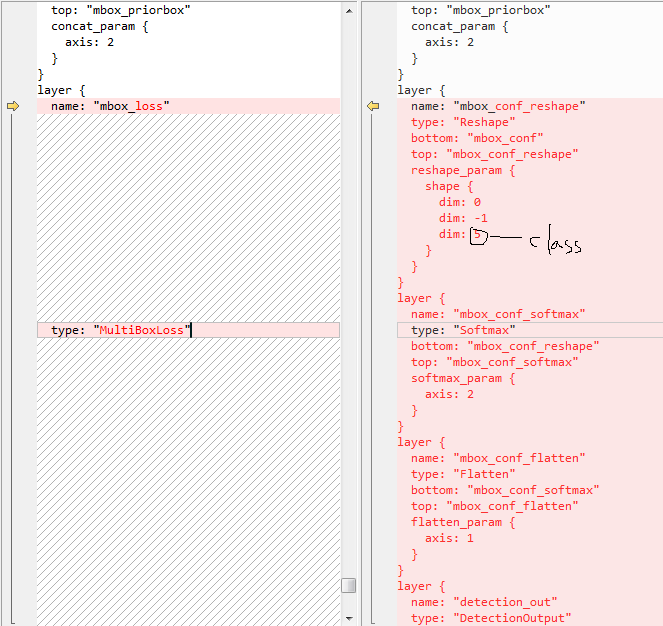
按照下面[train.prototxt与test.prototxt的不同](#_train.prototxt与test.prototxt对比结果如下：)，将train.prototxt改成test.prototxt。

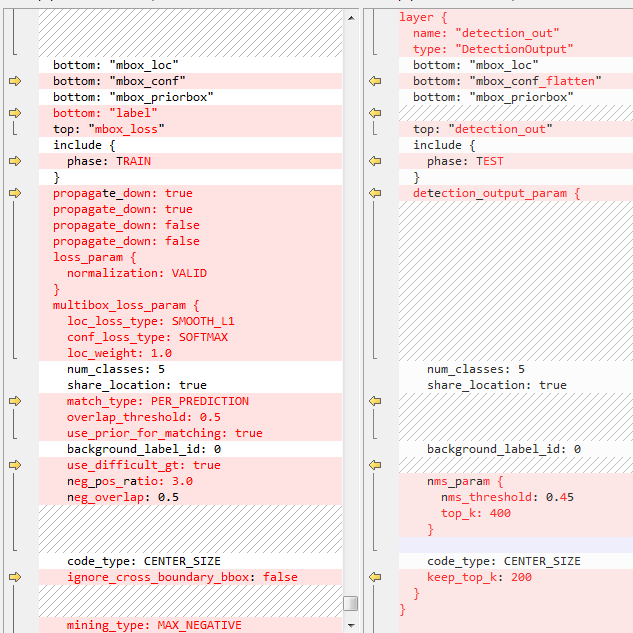
#### train.prototxt与test.prototxt对比结果如下：

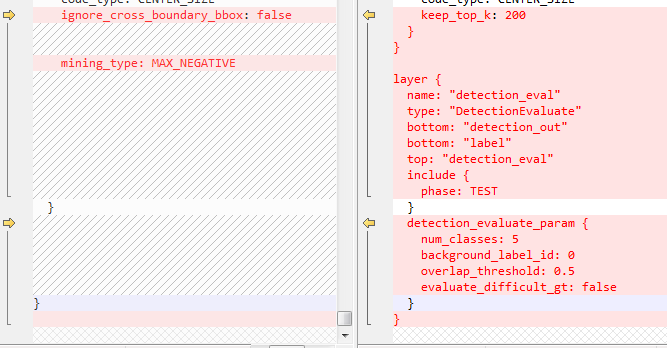












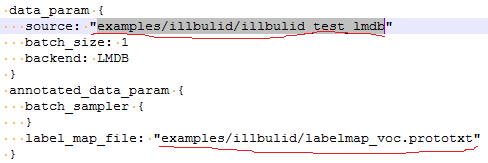
**备注：** 注意修改测试数据的路径，别的修改的地方在训练时都已经修改。

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。

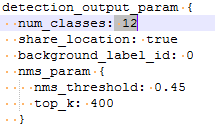
### 修改1：加载数据路径、标签索引文件路径

**备注：**对于测试模型，batch\_size=1

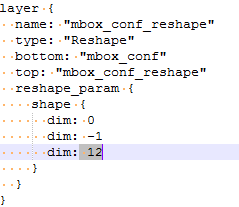


### 修改2：类别个数

**dim：** **12** （12为类别数目）



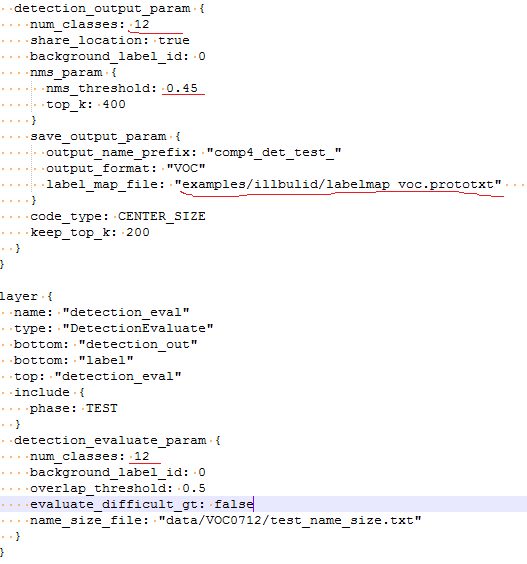
### 修改3：reshape维度



### 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

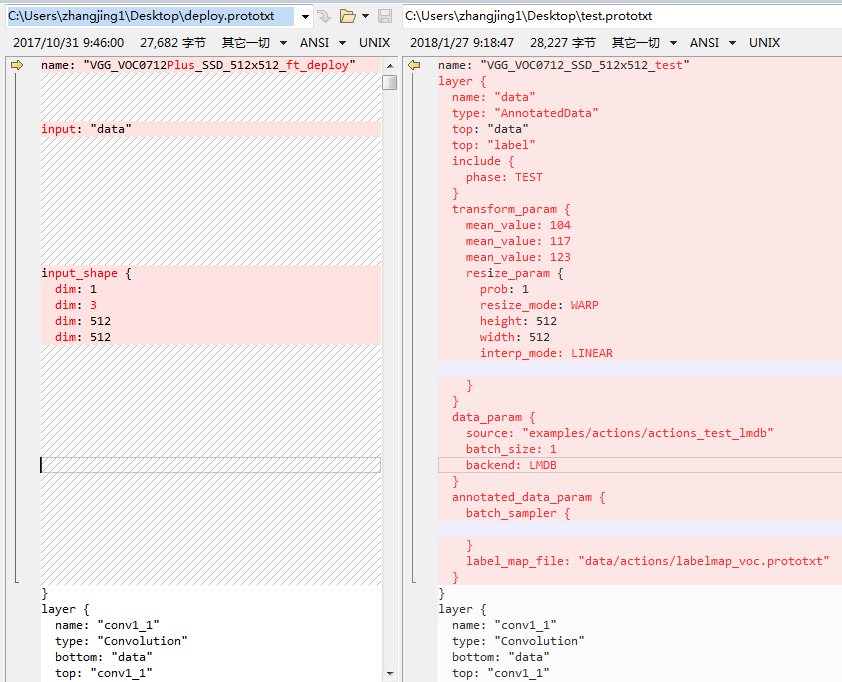
### 新建deploy.prototxt文件

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

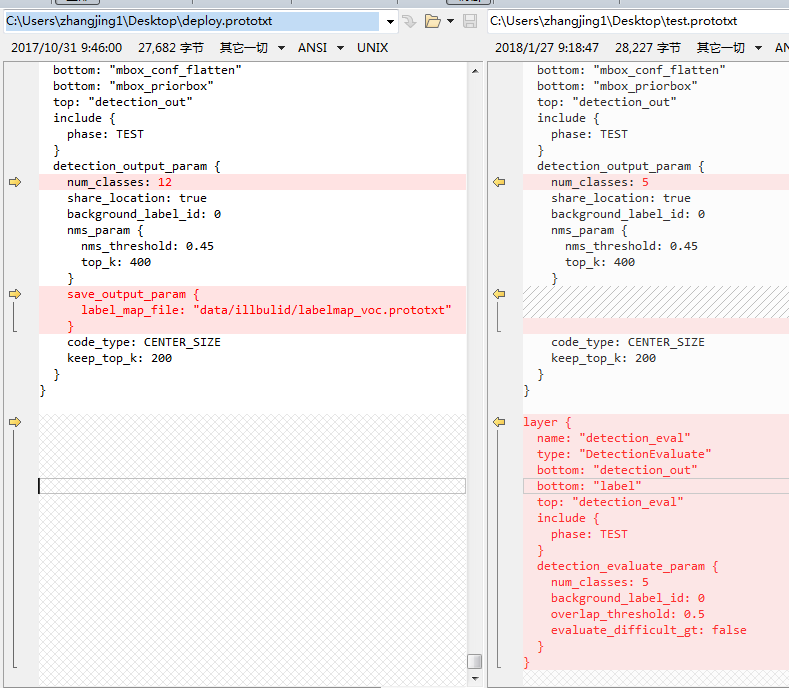
##### data层不同

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



##### 最后层不同

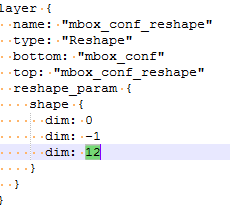
**最后层不同，deploy.prototxt与test.prototxt对比如下：**

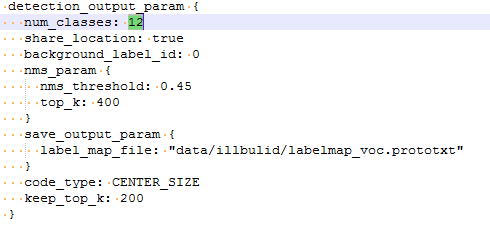


**备注：**save\_output\_param参数可以没有。

### 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，最后一层是MultiBoxLoss层；

test.prototxt: 测试，需要计算准确率, 多了DetectionOutput层、DetectionEvaluate层，少了

一些训练用的；**相比train.prototxt，test.prototxt少了MultiBoxLoss层，多了DetectionOutput层、DetectionEvaluate层、AnnotatedData层少了一些数据增强的参数。**

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

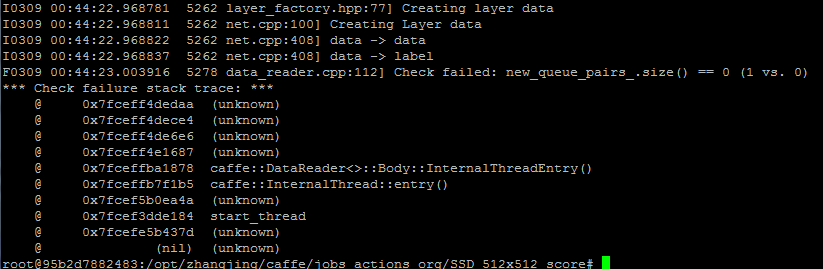
片，该网络结构用于测试。并且**比test.prototxt少“detection\_eval”层**。比

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)

## 6.error

1.执行eval.sh， 错误如下：

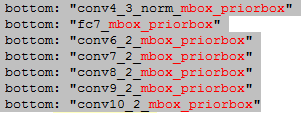


**原因**：train.prototxt使用"examples/actions\_new/actions\_new\_train\_lmdb"数据，test.prototxt测试训练集也是用"examples/actions\_new/actions\_new\_train\_lmdb"数据，应该使用"examples/actions\_new/actions\_new\_trainval\_lmdb"数据就OK.

# 其它知识点

## SDD使用的VGG16

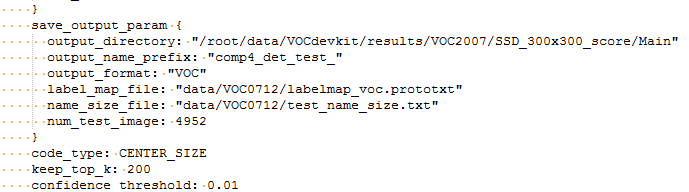
通过这7层来做预测：



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，例如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

少次后进行模型保存。

**snapshot\_prefix:** “examples/mnist/ VGG\_SSD\_512x512” #存放保存模型的位置以及模型

名称的前缀，**VGG\_SSD\_512x512**\_iter\_180000.solverstate

**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.

### 5.4 blob的存储

Blob 是 Caffe 中处理和传递实际数据的数据封装包，并且在 CPU 与 GPU 之间具有同步处理能力。从数学意义上说，blob是按C 风格连续存储的 N维数组。

Caffe基于blobs存储和交换数据，例如批量图像数据、模型参数以及用来进行优化的导数。

对于批量图像数据来说，blob常规的维数为图像数量N \*通道数K \*图像高度 H \*图像宽度W。Blob按行为主（row-major）进行存储，所以一个4 维blob中，坐标为(n, k, h, w)的值的物理位置为**（ （n \* K + k) \* H + h) \* W + w**，这也使得最后面/最右边的维度更新最快。

例如：图像大小（H, W），坐标 （h,w）值的物理位置为h\*W+w;

图像大小（K,H, W），坐标 （k,h,w）值的物理位置为(k\*H+h)\*W+w;

图像大小（N,K,H, W），坐标 （n,k,h,w）值的物理位置为((n\*K+k)\*H+h)\*W+w;

对于 blob 中的数据，我们关心的是 values（值）和 gradients（梯度） ，所以一个 blob单元存储了两块数据——data和diff。前者是我们在网络中传送的普通数据，后者是通过网络计算得到的梯度。 而且，由于数据既可存储在 CPU 上，也可存储在 GPU 上，因而有两种数据访问方式：静态方式，不改变数值；动态方式，改变数值。

const Dtype\* cpu\_data() const;

Dtype\* mutable\_cpu\_data();

### 5.5 MultiBoxLossLayer参数说明

multibox\_loss\_param {

loc\_loss\_type: SMOOTH\_L1 **#loc\_ loss，L1=|y\_new-y|, L2=(y\_new-y)^2)**

conf\_loss\_type: SOFTMAX

loc\_weight: 1.0 **# loss =(conf\_loss + loc\_weight\*loc\_loss)/batch\_size**

num\_classes: 21 **#num\_classes=目标个数+背景**

share\_location: true **#位置共享，如果为true，边框在不同的类中共享**

match\_type: PER\_PREDICTION **#在util/bbox\_util.cpp,训练中的匹配方法。定义为枚举，有**

**BIPARTITE和PER\_PREDICTION两种。如果match\_type为PER\_PREDICTION（即每张图预测），则使用overlap\_threshold来确定额外的匹配bbox。**

overlap\_threshold: 0.5 **#表示default box和ground truth重合度超过该阈值则为正样本,**

**阀值大小。即我们所说的IoU的大小**

use\_prior\_for\_matching: true #是否使用先验匹配，一般为true。

background\_label\_id: 0 #**背景标签的类别编号，一般为0**

use\_difficult\_gt: true **#是否考虑困难的ground truth，默认为true。**

neg\_pos\_ratio: 3.0 **# Hard Negative Mining策略中负正样本比例**

neg\_overlap: 0.5（**即如果重叠小于0.5则定义为负样本，Faster R-CNN设置为0.3**）

code\_type: CENTER\_SIZE # **#bbox的编码方式。此参数定义在PriorBoxParameter参数定**

**义解释中，为枚举类型，三种类型为：CORNER，CENTER\_SIZE和CORNER\_SIZE。**

ignore\_cross\_boundary\_bbox: false **#如果为true，则在匹配期间忽略跨边界bbox。 跨界**

**bbox是一个在图像区域之外的bbox。即将超出图像的预测边框剔除，这里我们不踢除，否则特征图边界点产生的先验框就没有任何意义。**

mining\_type: MAX\_NEGATIVE # **#训练期间的挖掘类型。定义为枚举，分别为三种类型： 若**

**为NONE则表示什么都不使用，这样会导致正负样本的严重不均衡；若为MAX\_NEGATIVE则根据分数选择底片；若为HARD\_EXAMPLE则选择基于“在线硬示例挖掘的基于训练区域的对象探测器”的硬实例，此类型即为SSD原文中所使用的Hard\_negative\_mining(负硬挖掘)策略。**

}

loss\_param = {   ###存储由损失层共享的参数的消息

 'normalization': VALID    #如何规范跨越批次，空间维度或其他维度聚集的损失层的损失。 目前只在

SoftmaxWithLoss和SigmoidCrossEntropyLoss图层中实现。按照批次中的示例数量

乘以空间维度。 在计算归一化因子时，不会忽略接收忽略标签的输出。定义为枚

举，四种类型分别是：FULL，除以不带ignore\_label的输出位置总数。 如果未设

置ignore\_label，则表现为FULL；VALID；BATCH\_SIZE，除以批量大小；NONE，不

要规范化损失。

   }

<http://blog.csdn.net/xunan003/article/details/79089280> （好多参数说明）

### 5.6 detection output layer

layer {

name: "detection\_out"

type: **"DetectionOutput"**

bottom: "mbox\_loc"

bottom: "mbox\_conf\_flatten"

bottom: "mbox\_priorbox"

top: "detection\_out"

include {

phase: **TEST**

}

detection\_output\_param {

num\_classes: 5 #类别数目 = 目标+背景

share\_location: true #位置共享

background\_label\_id: 0 #背景类别编号，这里为0

confidence\_threshold: 0.01 #只考虑可信度大于阈值的检测。 如果没有提供，请考虑所

有的框。

nms\_param {

nms\_threshold: 0.45

top\_k: 400

} #文中介绍，非最大抑制的作用就是消除多余的框，就是使评分低的框剔除。先选择置信度大于confidence\_threshold的前top\_k个结果，再进行非极大值抑制。

code\_type: CENTER\_SIZE

keep\_top\_k: 200 #nms步之后每个图像要保留的bbox总数。-1表示在nms步之后保留所有的bbox。

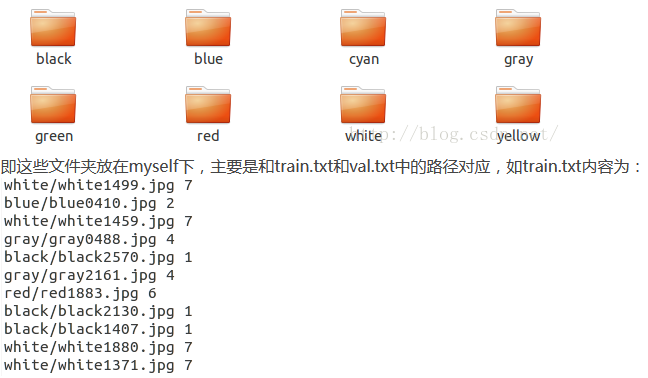
}

}

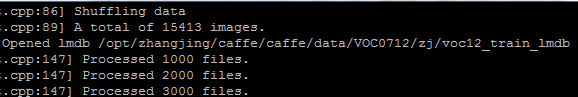
## 6. ****create\_imagenet.sh生成lmdb****

### 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。**

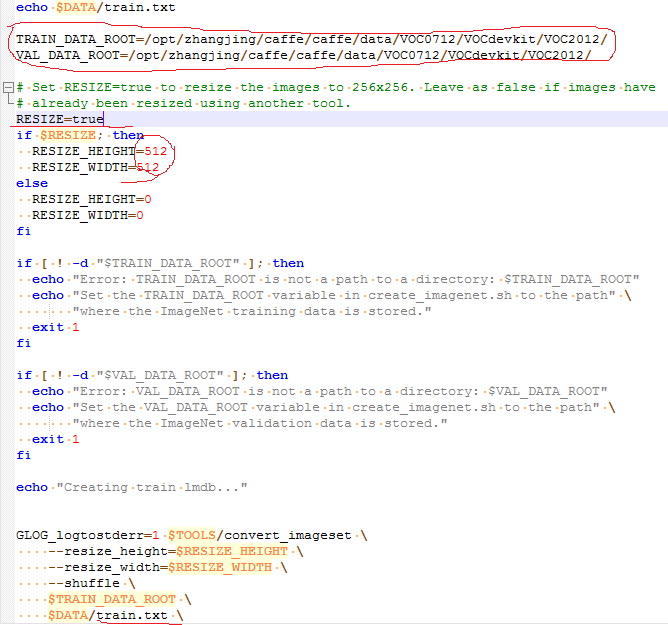
****

**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

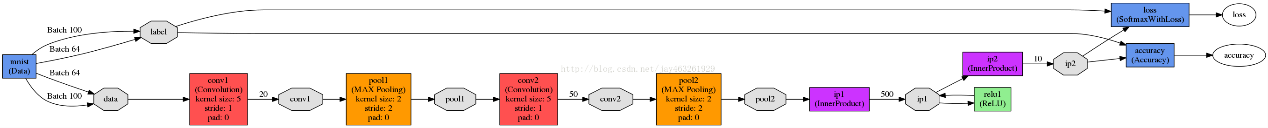
### 代码

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)))

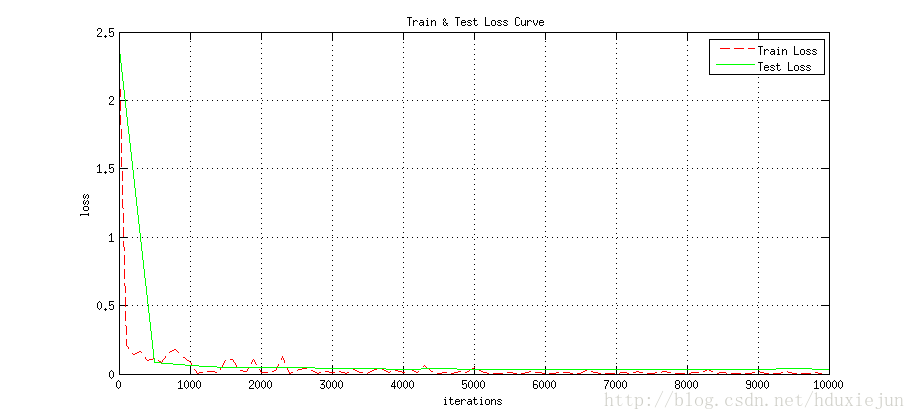
if \_\_name\_\_ == '\_\_main\_\_':

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、768\*768）

**备注**：改大输入图像大小后测试mAP时，会报内存不足；解决办法是将train.prototxt的输入图像大小改小，test.prototxt的输入图像大小不变。

## 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。

# 代码执行流程

<http://blog.sina.com.cn/s/blog_9a3e0f110102w9r1.html>

caffe调用：

caffe/tools/caffe.cpp（main函数）->

# 代码备份

## test\_512.py代码（预测代码）

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')

# 原理解释

## ****算法概述：****

SSD，全称Single Shot MultiBox Detector，是**Wei Liu**在**ECCV 2016**上提出的一种目标检测算法，截止目前是主要的检测框架之一，相比Faster RCNN有明显的速度优势，相比YOLO又有明显的mAP优势（不过已经被CVPR 2017d的YOLO9000超越）。

**SSD算法是最近一年比较优秀的object detection算法，主要特点在于采用了特征融合。**

**该算法是一种直接预测bounding box的坐标和类别的object detection算法，没有生成proposal的过程**。针对不同大小的物体检测，传统的做法是将图像转换成不同的大小，然后分别处理，最后将结果综合起来，而**本文的ssd利用不同卷积层的feature map进行综合也能达到同样的效果**。算法的主网络结构是VGG16，将两个全连接层改成卷积层再增加4个卷积层构造网络结构。对其中5个不同的卷积层的输出分别用两个3\*3的卷积核进行卷积，**一个输出分类用的confidence，每个default box生成21个confidence（这是针对VOC数据集包含20个object类别而言的）；一个输出回归用的localization，每个default box生成4个坐标值（x，y，w，h）**。另外这5个卷积层还经过priorBox层生成default box（生成的是坐标）。上面所述的5个卷积层中每一层的default box的数量是给定的。最后将前面三个计算结果分别合并然后传递给loss层。

**算法的结果：对于300\*300的输入，SSD可以在VOC2007 test上有74.3%的mAP，速度是59 FPS(Nvidia Titan X)，对于512\*512的输入, SSD可以有76.9%的mAP**。相比之下Faster RCNN是73.2%的mAP和7FPS，YOLO是63.4%的mAP和45FPS。即便对于分辨率较低的输入也能取得较高的准确率。可见作者并非像传统的做法一样以牺牲准确率的方式来提高检测速度。**作者认为自己的算法之所以在速度上有明显的提升，得益于去掉了bounding box proposal以及后续的pixel或feature的resampling步骤。**

YOLO能达到实时的效果，但**YOLO有一些缺陷**：每个网格只预测一个物体，容易造成漏检；对于物体的尺度相对比较敏感，对于尺度变化较大的物体泛化能力较差。针对 YOLO 中的这些不足，该论文提出的方法 SSD 在这两方面都有所改进，同时兼顾了 mAP 和实时性的要求。在满足实时性的条件下，接近 state of art 的结果。作者的**思路**就是Faster R-CNN+YOLO，利用YOLO的思路和Faster R-CNN的anchor box的思想。

**参考文献**：<https://zhuanlan.zhihu.com/p/24954433?refer=xiaoleimlnote>

<https://www.cnblogs.com/demian/p/7576046.html>

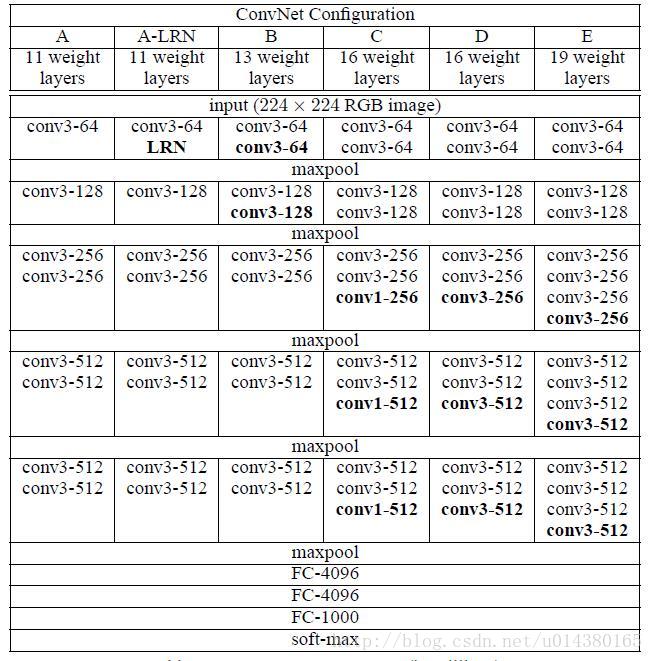
<http://blog.csdn.net/u014380165/article/details/72824889> （还不错）

<https://arxiv.org/pdf/1512.02325.pdf> （论文地址）

## ****算法详解：****

SSD算法在训练的时候只需要一张输入图像及其每个object的ground truth boxes。

基本的网络结构是基于VGG16，在ImageNet数据集上预训练完以后用两个新的卷积层代替fc6和fc7，另外对pool5也做了一点小改动（**其它池化是kernel\_size=2，stride=2，池化后feature map宽高度为缩小一半；而pool5是kernel\_size=3，stride=1，pad=1，池化后feature map宽高度不变**），还增加了4或5个卷积层构成本文的网络。VGG的结构如下图所示：

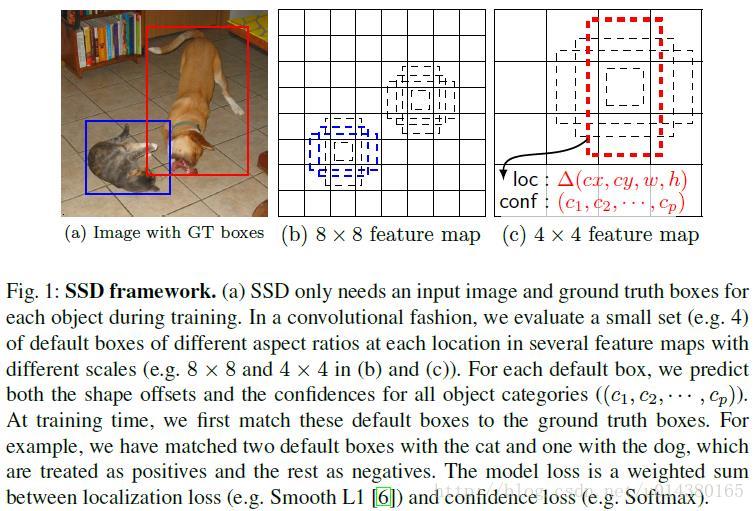


**文章的一个核心是作者同时采用lower和upper的feature maps做检测。**如下图Fig1，有8\*8和4\*4两种大小的feature maps，而feature map cell就是其中的每一个小格。另外有一个概念：**default box，是指在feature map的每个小格(cell)上都有一系列固定大小的box**，如下图有4个（下图中的虚线框，仔细看格子的中间有比格子还小的一个box）。假设每个feature map cell有k个default box，那么对于每个default box都需要预测c个类别score和4个offset，那么如果一个feature map的大小是m\*n，也就是有m\*n个feature map cell，那么这个feature map就一共有（c+4）\*k\*m\*n个输出（**这些输出个数的含义是：采用3\*3的卷积核对该层的feature map卷积时卷积核的个数，包含两部分（实际code是分别用不同数量的3\*3卷积核对该层feature map进行卷积），数量c\*k\*m\*n是confidence输出，表示每个default box的confidence，就是类别；数量4\*k\*m\*n是localization输出，表示每个default box的坐标**）。作者的实验也表明default box的shape数量越多，效果越好。

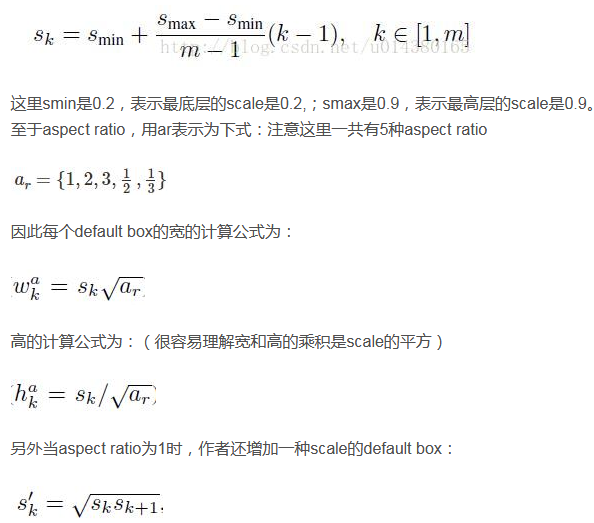
所以这里用到的default box和Faster RCNN中的anchor很像，在Faster RCNN中anchor只用在最后一个卷积层，但是在本文中，default box是应用在多个不同层的feature map上。

**下图还有一个重要的信息是：在训练阶段，算法在一开始会先将这些default box和ground truth box进行匹配（IOU>0.5,可在prototxt中设置），比如蓝色的两个虚线框和猫的ground truth box匹配上了，一个红色的虚线框和狗的ground truth box匹配上了。所以一个ground truth可能对应多个default box。在预测阶段，直接预测每个default box的偏移以及对每个类别相应的得分，最后通过NMS得到最终的结果。**

Fig1（c）说明对于每个default box，同时预测它的坐标offset和所有类的confidence。



那么default box的scale（大小）和aspect ratio（宽高比）要怎么定呢？假设我们用**m个feature maps做预测**，那么对于每个featuer map而言其default box的scale是按以下公式计算的：

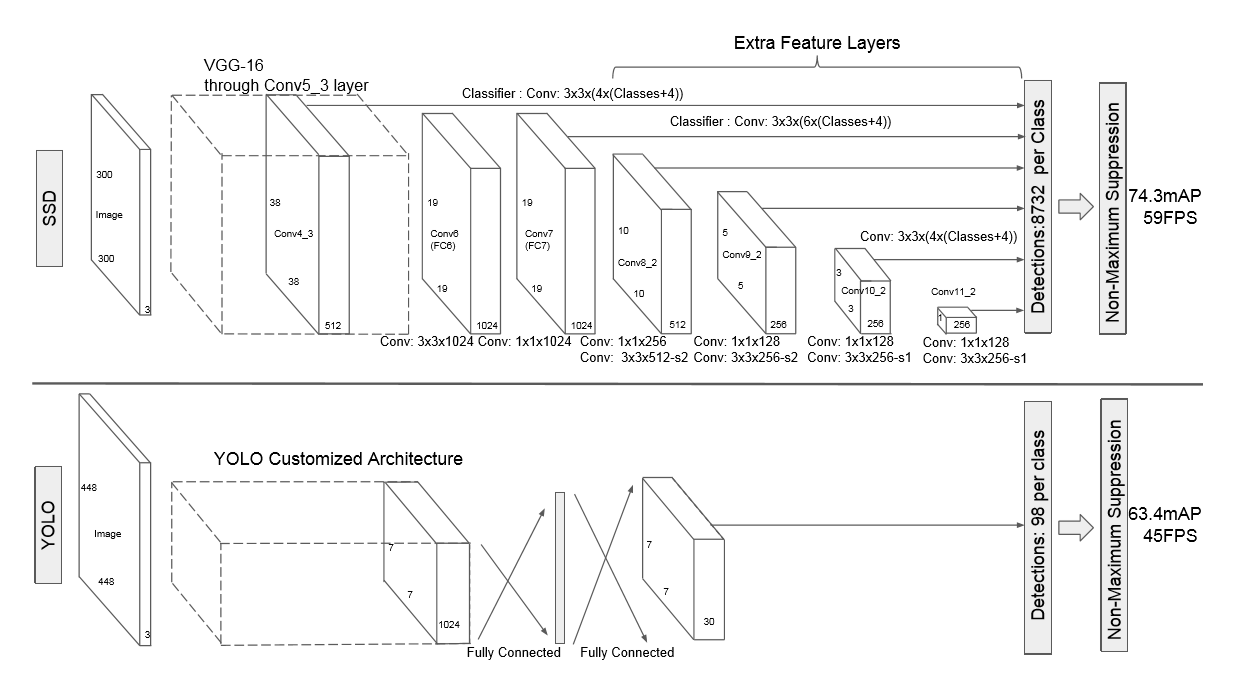


因此，**对于每个feature map cell而言，一共有6种default box。**

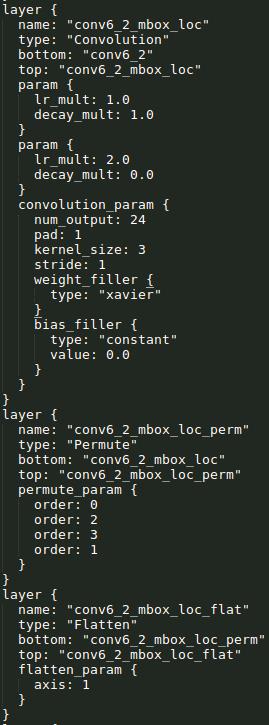
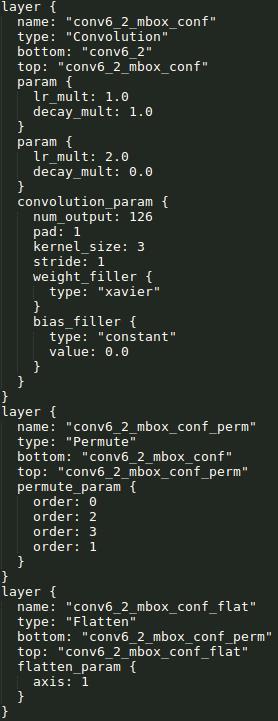
**可以看出这种default box在不同的feature层有不同的scale，在同一个feature层又有不同的aspect ratio，因此基本上可以覆盖输入图像中的各种形状和大小的object！**

显然，当default box和grount truth匹配上了，那么这个default box就是positive example（正样本），如果匹配不上，就是negative example（负样本），显然这样产生的负样本的数量要远远多于正样本。于是作者将负样本按照confidence loss进行排序，然后选择排名靠前的一些负样本作为训练，使得最后**负样本和正样本的比例在3:1**左右。

下图是**SSD算法和YOLO算法的结构图对比**。YOLO算法的输入是**448\*448\*3**，输出是**7\*7\*30**，这7\*7个grid cell**一共预测98个bounding box**。SSD算法是在原来VGG16的后面添加了几个卷积层来预测offset和confidence（相比之下YOLO算法是采用**全连接层**），算法的输入是300\*300\*3，采用conv4\_3，conv7，conv8\_2，conv9\_2，conv10\_2和conv11\_2的输出来预测location和confidence。

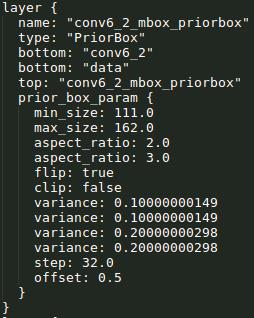


**详细讲一下SSD的结构，可以参看Caffe代码。SSD的结构为conv1\_1，conv1\_2，conv2\_1，conv2\_2，conv3\_1，conv3\_2，conv3\_3，conv4\_1，conv4\_2，conv4\_3，conv5\_1，conv5\_2，conv5\_3（512），fc6：3\*3\*1024的卷积（原来VGG16中的fc6是全连接层，这里变成卷积层，下面的fc7层同理），fc7:1\*1\*1024的卷积，conv6\_1，conv6\_2（对应上图的conv8\_2），……，conv9\_1，conv9\_2，loss。然后针对conv4\_3（4），fc7（6），conv6\_2（6），conv7\_2（6），conv8\_2（4），conv9\_2（4）的每一个再分别采用两个3\*3大小的卷积核进行卷积，这两个卷积核是并列的（括号里的数字代表default box的数量，可以参考Caffe代码，所以上图中SSD结构的倒数第二列的数字8732表示的是所有default box的数量，是这么来的38\*38\*4+19\*19\*6+10\*10\*6+5\*5\*6+3\*3\*4+1\*1\*4=8732），这两个3\*3的卷积核一个是用来做localization的（回归用，如果default box是6个，那么就有6\*4=24个这样的卷积核，卷积后map的大小和卷积前一样，因为pad=1，下同），另一个是用来做confidence的（分类用，如果default box是6个，VOC的object类别有20个，那么就有6\*（20+1）=126个这样的卷积核）。如下图是conv6\_2的localizaiton的3\*3卷积核操作，卷积核个数是24（6\*4=24，由于pad=1，所以卷积结果的map大小不变，下同）：这里的permute层就是交换的作用，比如你卷积后的维度是32\*24\*19\*19，那么经过交换层后就变成32\*19\*19\*24，顺序变了而已。而flatten层的作用就是将32\*19\*19\*24变成32\*8664，32是batchsize的大小。**

位置：卷积核个数是126（6\*21=126） 类别：卷积核个数是24（6\*4=24）

**然后是一个生成default box的操作，根据最小尺寸，最大尺寸以及宽高比来生成，step表示该层的一个像素点相当于最开始输入图像的1/32，简单讲就是感受野，源码里面是通过将原始的input image的大小除以该层feature map的大小来得到的。variance目测是一个尺度变换，本文的四个坐标采用的是中心坐标加上长宽，计算loss的时候可能需要对中心坐标的loss和长宽的loss做一个权衡，所以有了这个variance。如果采用的是box的四大顶点坐标这种方式，默认variance都是0.1，即相互之间没有权重差异。经过上述3个操作后，对这一层feature的处理就结束了。**



**备注：一般VGG16的train.prototxt中使用**

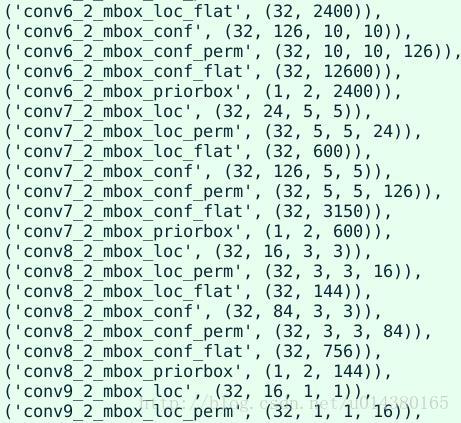
variance: 0.1

variance: 0.1

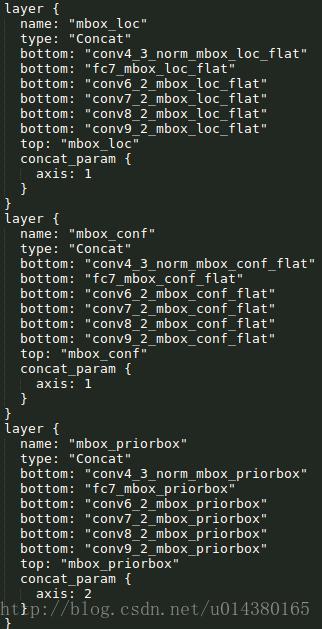
variance: 0.2

variance: 0.2

稍微看下几个层的输出维度，注意看priorbox的维度，以conv8\_2\_mbox\_priorbox为例，是（1,2,144），这个144表示生成的default box的所有坐标，所以和前面回归的坐标个数是一样的：3\*3\*4\*4（feature map大小3\*3，default box num=4, 4=(dx,dy,dw,dh)）。2是和variance相关的。



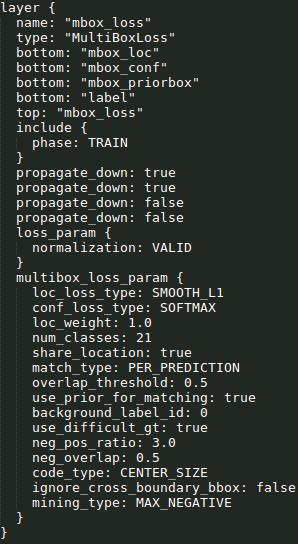
对前面所列的5个卷积层输出都执行上述的操作后，就将得到的结果合并：**采用Concat，类似googleNet的Inception操作，是通道合并而不是数值相加**。



这是几个通道合并后的维度：

这里写图片描述

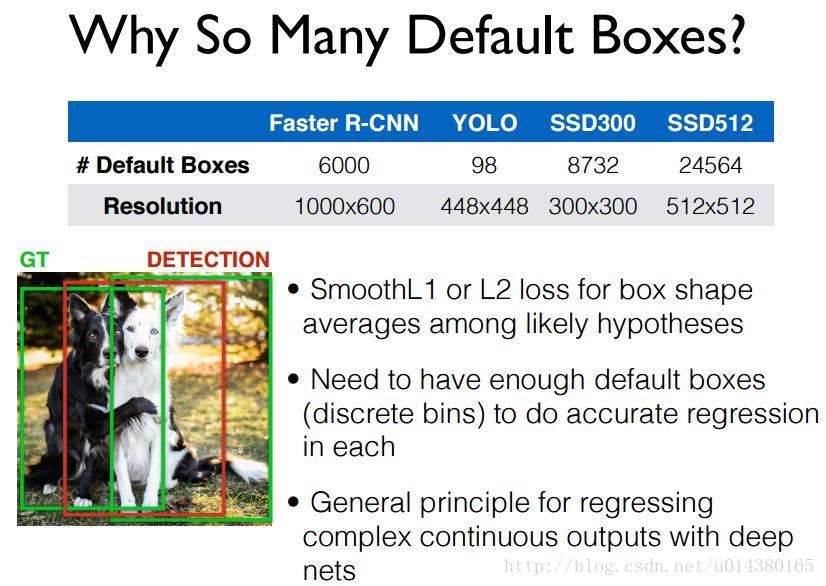
最后就是作者自定义的损失函数层，这里的**overlap\_threshold**表示default box和ground truth的重合度超过这个阈值则为正样本。另外我觉得具体哪些default box是正样本，哪些是负样本是在loss层计算出来的，不过这个细节与算法关系不大：



**损失函数方面**：和Faster RCNN的基本一样，由分类和回归两部分组成，可以参考Faster RCNN，这里不细讲。总之，回归部分的loss是希望预测的box和default box的差距尽可能跟ground truth和default box的差距接近，这样预测的box就能尽量和ground truth一样。

这里写图片描述

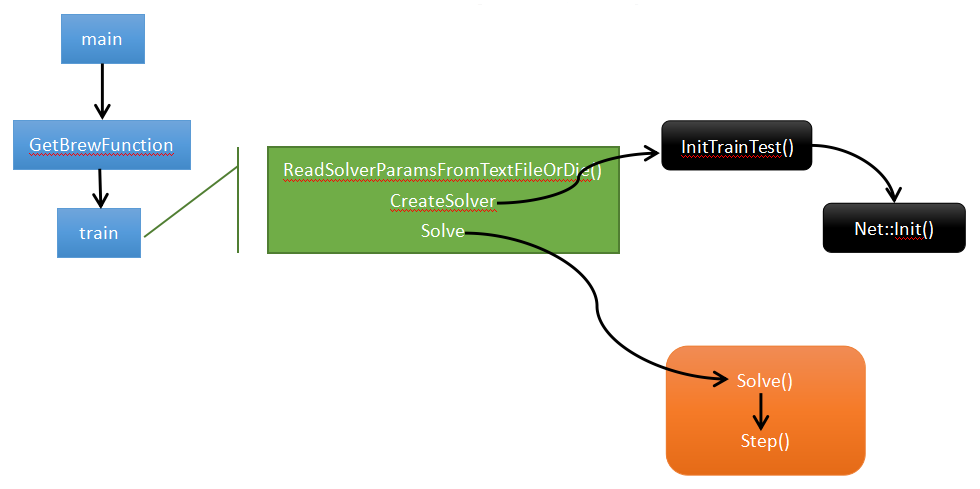
这里稍微列了下几种object detection算法的default boxes数量以及为什么要有这么多的box：



**文中作者提到该算法对于小的object的detection比大的object要差。作者认为原因在于这些小的object在网络的顶层所占的信息量太少，所以增加输入图像的尺寸对于小的object的检测有帮助。另外增加数据集对于小的object的检测也有帮助，原因在于随机裁剪后的图像相当于“放大”原图像，所以这样的裁剪操作不仅增加了图像数量，也放大了图像。**

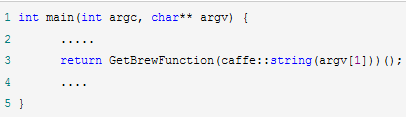
## Caffe的代码整体流程

### 整体流程图



Caffe的整体流程图

### 程序入口：main()



main函数在**tools/caffe.cpp**中，g\_brew\_map实现过程，首先通过 typedef定义函数指针 typedef int (\*BrewFunction)(); 这个是用typedef定义函数指针方法。这个程序定义一个BrewFunction函数指针类型，在caffe.cpp 中 BrewFunction 作为GetBrewFunction()函数的返回类型，可以是 train()，test()，device\_query()，time() 这四个函数指针的其中一个。在train()，test()，中可以调用solver类的函数，从而进入到net，进入到每一层，运行整个caffe程序。然后对每个函数注册。

1 RegisterBrewFunction(train) #训练或者调整一个模型

2 RegisterBrewFunction(test) #在测试集上测试一个模型

3 RegisterBrewFunction(device\_query) #打印GPU的调试信息

4 RegisterBrewFunction(time) #测一个模型的执行时间

如果需要，可以增加其他的方式，然后通过RegisterBrewFunction()函数注册一下即可。

#### train()

接着调用train()函数，train函数中主要有三个方法**ReadSolverParamsFromTextFileOrDie、CreateSolver、Solve。**

int train()

{

3 ......

4 caffe::SolverParameter solver\_param;

5 caffe::**ReadSolverParamsFromTextFileOrDie**(FLAGS\_solver, &solver\_param);//**从-solver参数读取solver\_param**

6 ......

7 shared\_ptr<caffe::Solver<float> >

8 solver(caffe::SolverRegistry<float>::**CreateSolver**(solver\_param));//**根据参数创建solver**，同样采用string到函数指针的映射实现，用到了工厂模式

10 ......

17 if (gpus.size() > 1)

{

18 caffe::P2PSync<float> sync(solver, NULL, solver->param());

19 sync.Run(gpus);

20 }

else

{

21 LOG(INFO) << "Starting Optimization";

22 solver->**Solve**();//**开始训练网络**

23 }

24 LOG(INFO) << "Optimization Done.";

25 return 0;

26 }

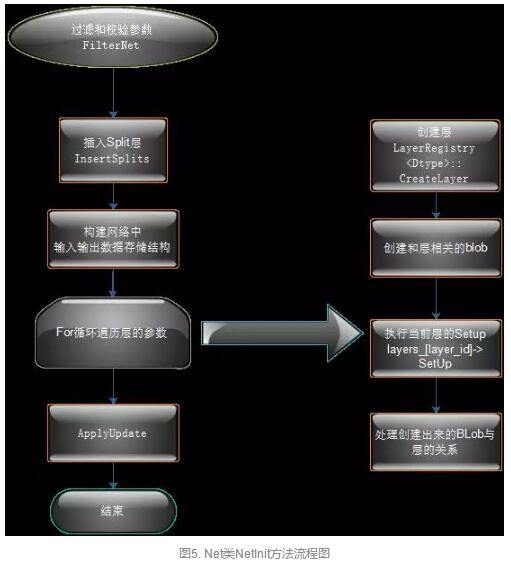
##### CreateSolver()

CreateSolver函数构建solver和net，该函数是初始化的入口，会通过执行Solver的构造函数，调用void Solver<Dtype>::Init(const SolverParameter& param)，该函数内有InitTrainNet()、InitTestNets()。对于InitTrainNet函数：

......

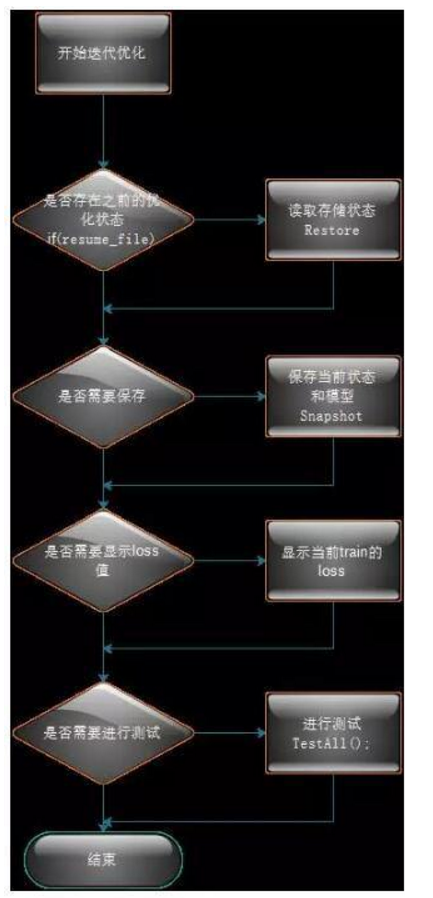
net\_.reset(new Net<Dtype>(net\_param));

调用Net类的构造函数，然后执行Init()操作，该函数具体的内容如下图所示：

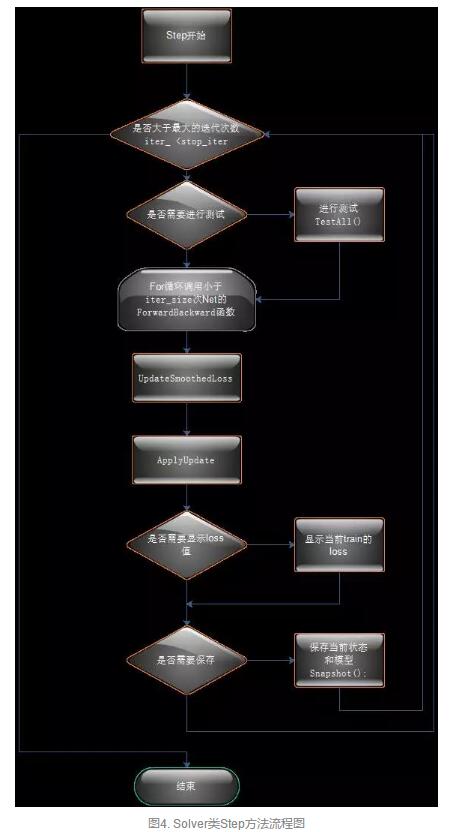


##### Solver() # src/solver.cpp

初始化的总体流程大概就是新建一个Solver对象，然后调用Solver类的构造函数，然后在Solver的构造函数中又会新建Net类实例，在Net类的构造函数中又会新建各个layer的实例，一直具体到设置每个Blob，大概就完成了网络初始化的工作了。



###### 执行Step函数



**备注：** ApplyUpdate();// BP算法更新权重

while循环中先调用了网络类Net::ForwardBackward()成员函数进行正向传导和反向传导，并计算loss. (**备注**：ForwardBackward()在include/net.hpp中)

[复制代码](javascript:void(0);)

1 Dtype ForwardBackward() {

2 Dtype loss;

3 //正向传导

4 Forward(&loss);

5 //反向传导

6 Backward();

7 return loss;

8 }

[复制代码](javascript:void(0);)

而Fordward函数中调用了ForwardFromTo，而FordwardFromTo又调用了每个layer的Fordward。反向传导函数Backward()调用了BackwardFromTo(int start, int end)函数。正向传导和反向传导结束后，再调用**SGDSolver::ApplyUpdate()成员函数进行权重更新**。

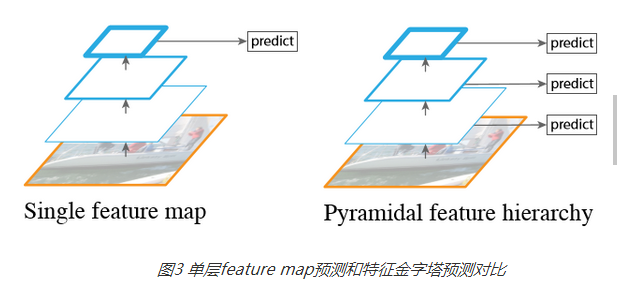
* ForwardBackward：按顺序调用了Forward和Backward。
* ForwardFromTo(int start, int end)：执行从start层到end层的前向传递，采用简单的for循环调用。**forward只要计算损失loss;**
* BackwardFromTo(int start, int end)：和前面的ForwardFromTo函数类似，调用从start层到end层的反向传递。**backward主要根据loss来计算梯度**，caffe通过自动求导并反向组合每一层的梯度来计算整个网络的梯度。
* ToProto函数完成网络的序列化到文件，循环调用了每个层的ToProto函数，（在net.cpp）

**参考文献：**<https://www.cnblogs.com/liuzhongfeng/p/7289956.html>

## 知识备注

### 特点1：特征金字塔结构分类回归

从yolo与ssd的结构图对比，可以看出YOLO在卷积层后接全连接层，即检测时只利用了最高层feature maps(包括Faster RCNN也是如此)；而SSD采用了**特征金字塔结构进行检测**，即检测时利用了conv4\_3、FC7、conv6\_2、conv7\_2、conv8\_2、conv9\_2、conv10\_2这些大小不同的feature maps，**在多个feature maps上同时进行softmax分类和位置回归，**如下图所示：

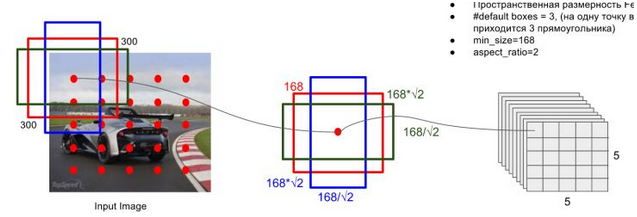


在低层的feature map,感受野比较小，高层的感受野比较大，在不同的feature map进行卷积，可以达到多尺度的目的。SSD最低层的feature\_map是conv4\_3，所以对小目标检测效果不好。

### 特点2：Prior Box

#### priorbox如何产生？

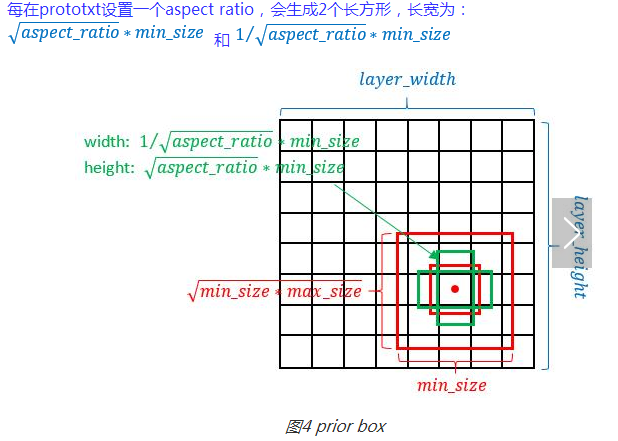
在5\*5\*256的feature map上 anchor的示例：**（scale=168(针对原图说的)，宽高比=1:1,1:2,2:1）**



假如feature map 的size 为 m\*n, 通道数为 p，使用的卷积核大小为 3\*3\*p。每个 feature map 上的每个特征点对应 k 个 default boxes，物体的类别数为 c，那么一个feature map就需要使用 k(c+4)个这样的卷积滤波器，最后有 (m\*n) \*k\* (c+4)个输出。

在SSD中引入了Prior Box，实际上与anchor非常类似，就是一些目标的**预选框**，后续通过softmax分类+bounding box regression获得真实目标的位置。SSD按照如下规则生成prior box：

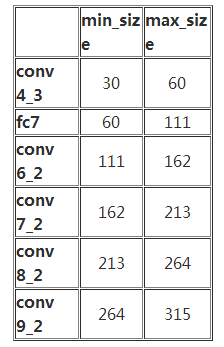
以feature map上**每个点的中点**为中心（offset=0.5），生成一系列同心的prior box（然后中心点的坐标会乘以**step**，相当于从feature map位置映射回原图位置）。prior box最小边长为min\_size，最大边长为。



而每个feature map对应prior box的min\_size和max\_size由以下公式决定，公式中**m是使用feature map的数量**（SSD 300中m=6）：

C:\Users\Administrator\Desktop\DT$6~0AAN3`AT(6(WOKGPC8.png

第一层feature map对应的min\_size=S1，max\_size=S2；第二层min\_size=S2，max\_size=S3；其他类推。



不过依然可以看出，**SSD使用低层feature map检测小目标，使用高层feature map检测大目标，这也应该是SSD的突出贡献了。**其中SSD 300在conv4\_3生成prior box的conv4\_3\_norm\_priorbox层prototxt定义如下：

[cpp] view plain copy

layer {

  name: "conv4\_3\_norm\_mbox\_priorbox"

  type: "PriorBox"

  bottom: "conv4\_3\_norm"

  bottom: "data"

  top: "conv4\_3\_norm\_mbox\_priorbox"

  prior\_box\_param {

**min\_size: 30.0**

**max\_size: 60.0**

**aspect\_ratio: 2**

**flip: true**

    clip: false

    variance: 0.1

    variance: 0.1

    variance: 0.2

    variance: 0.2

    step: 8

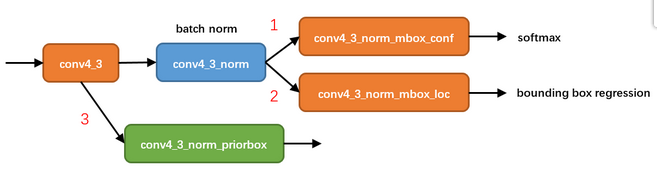
    offset: 0.5

  }

}

#### prior box如何使用？

知道了priorbox如何产生，接下来分析prior box如何使用。这里以conv4\_3为例进行分析。



从图可以看到，在conv4\_3 feature map网络pipeline分为了3条线路：

1. 一次batch norm+一次卷积后，生成了**[1, num\_class\*num\_priorbox, layer\_height, layer\_width]**大小的feature用于softmax分类目标和非目标（其中num\_class是目标类别，SSD 300中num\_class = 21)。
2. 一次batch norm+一次卷积后，生成了[1, 4\*num\_priorbox, layer\_height, layer\_width]大小的feature用于bounding box regression（即每个点一组[dxmin，dymin，dxmax，dymax]）。
3. 生成了[1, 2, 4\*num\_priorbox]大小的prior box blob，其中2个channel分别存储prior box的4个点坐标和对应的4个variance。

后续通过softmax分类+bounding box regression即**可从priox box中预测到目标**，熟悉Faster RCNN的读者应该对上述过程应该并不陌生。其实pribox box的与Faster RCNN中的anchor非常类似，**都是目标的预设框**，没有本质的差异。**区别是每个位置的prior box一般是4~6个，少于Faster RCNN默认的9个anchor**；同时**prior box是设置在不同尺度的feature maps上的，而且大小不同**。

缩进还有一个细节就是上面prototxt中的4个variance，这实际上是一种bounding regression中的权重。在图线路(2)中，网络输出[dxmin，dymin，dxmax，dymax]，即对应下面代码中bbox；然后利用如下方法进行针对prior box的位置回归：

decode\_bbox->set\_xmin(

    prior\_bbox.xmin() + prior\_variance[0] \* bbox.xmin() \* prior\_width);

decode\_bbox->set\_ymin(

    prior\_bbox.ymin() + prior\_variance[1] \* bbox.ymin() \* prior\_height);

decode\_bbox->set\_xmax(

    prior\_bbox.xmax() + prior\_variance[2] \* bbox.xmax() \* prior\_width);

decode\_bbox->set\_ymax(

    prior\_bbox.ymax() + prior\_variance[3] \* bbox.ymax() \* prior\_height);

上述代码可以在**SSD box\_utils.cpp的void DecodeBBox()**函数见到。

### 多个feature maps如何协同工作？

300是使用包括conv4\_3在内的共计6个feature maps一同检测出最终目标的。在网络运行的时候不会使用一个feature map单独计算一次softmax socre+box regression，否则训练时如何匹配ground truth都是问题。那么多个feature maps如何协同工作？这时候就要用到**Permute，Flatten和Concat这3种层了**。其中conv4\_3\_norm\_conf\_perm的prototxt定义如下：

#### Permute层：换位

layer {

  name: "conv4\_3\_norm\_mbox\_conf\_perm"

  type: **"Permute"**

  bottom: "conv4\_3\_norm\_mbox\_conf"

  top: "conv4\_3\_norm\_mbox\_conf\_perm"

  permute\_param {

    order: 0

    order: 2

    order: 3

    order: 1

  }

}

Permute是SSD中自带的层，上面conv4\_3\_norm\_mbox\_conf\_perm的的定义。Permute相当于交换caffe blob中的数据维度。在正常情况下caffe blob的顺序为：

bottom blob = **[batch\_num, channel, height, width]**

经过conv4\_3\_norm\_mbox\_conf\_perm后的caffe blob为：

top blob = **[batch\_num, height, width, channel]**

#### Flatten层：

layer {

name: "conv4\_3\_norm\_mbox\_conf\_flat"

type: **"Flatten"**

bottom: "conv4\_3\_norm\_mbox\_conf\_perm"

top: "conv4\_3\_norm\_mbox\_conf\_flat"

flatten\_param {

**axis: 1**

}

}

Flatten是SSD中自带的层，上面conv4\_3\_norm\_mbox\_conf\_flat的的定义。Flatten相当于改变caffe blob中数据的维度。在正常情况下caffe blob的顺序为：

bottom blob = **[batch\_num, height, width, channel]**

经过conv4\_3\_norm\_mbox\_conf\_flat后的caffe blob为：

top blob =**[batch\_num, height\*width\*channel** ]

#### Concat层：

layer {

name: "mbox\_loc"

type: **"Concat"**

bottom: "conv4\_3\_norm\_mbox\_loc\_flat"

bottom: "fc7\_mbox\_loc\_flat"

bottom: "conv6\_2\_mbox\_loc\_flat"

bottom: "conv7\_2\_mbox\_loc\_flat"

bottom: "conv8\_2\_mbox\_loc\_flat"

bottom: "conv9\_2\_mbox\_loc\_flat"

top: "mbox\_loc"

concat\_param {

**axis: 1**

}

}

layer {

name: "mbox\_conf"

type: **"Concat"**

bottom: "conv4\_3\_norm\_mbox\_conf\_flat"

bottom: "fc7\_mbox\_conf\_flat"

bottom: "conv6\_2\_mbox\_conf\_flat"

bottom: "conv7\_2\_mbox\_conf\_flat"

bottom: "conv8\_2\_mbox\_conf\_flat"

bottom: "conv9\_2\_mbox\_conf\_flat"

top: "mbox\_conf"

concat\_param {

**axis: 1**

}

}

layer {

name: "mbox\_priorbox"

type: **"Concat"**

bottom: "conv4\_3\_norm\_mbox\_priorbox"

bottom: "fc7\_mbox\_priorbox"

bottom: "conv6\_2\_mbox\_priorbox"

bottom: "conv7\_2\_mbox\_priorbox"

bottom: "conv8\_2\_mbox\_priorbox"

bottom: "conv9\_2\_mbox\_priorbox"

top: "mbox\_priorbox"

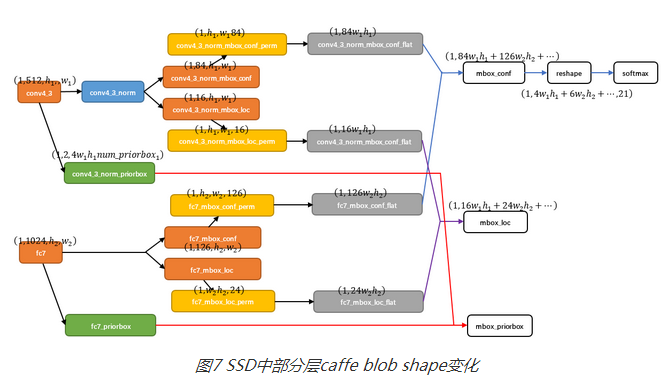
concat\_param {

**axis: 2**

}

}

#### caffe blob shape变化

****

那么接下来以conv4\_3和fc7为例分析SSD是如何将不同size的feature map组合在一起进行prediction。图7展示了conv4\_3和fc7合并在一起的过程中caffe blob shape变化（其他层类似，考虑到图片大小没有画出来，请脑补）。

对于conv4\_3 feature map，conv4\_3\_norm\_priorbox（priorbox层）设置了每个点共有**4个prior box**。由于SSD 300共有21个分类，所以conv4\_3\_norm\_mbox\_conf的channel值为num\_priorbox \* num\_class = 4 \* 21 = 84；而每个prior box都要回归出4个位置变换量，所以conv4\_3\_norm\_mbox\_loc的caffe blob channel值为4 \* 4 = 16。

fc7每个点有6个prior box，其他feature map同理。

经过一系列图7展示的caffe blob shape变化后，最后拼接成mbox\_conf和mbox\_loc。而mbox\_conf后接reshape，再进行softmax（**为何在softmax前进行reshape**，Faster RCNN有提及）。

最后这些值输出detection\_out\_layer，获得检测结果。

#### 为何mbox\_conf后、softmax前接reshape？？？

### 正负样本：

对于每个输入图像：

1. 对**每个标定**的ground truth（真实候选区域），与其重叠比例最大的default box记为**正**

**样本** ；

1. 对a)剩余的default box，如果其**与某个标定**重叠比例大于0.5（可在train.prototxt中

overlap\_threshold设置），记为**正样本**；如果其**与任意**一个标定的重叠比例都小于0.3（可在train.prototxt中neg\_overlap设置），记为**负样本**；

c. 对a),b)剩余的anchor，弃去不用。

d. 跨越图像边界的anchor弃去不用。

**备注：**其他的作为负样本（每个default box要么是正样本box要么是负样本box）。

### Hard Negative Mining：

用于预测的 feature map 上的每个点都对应有 6 个不同的 default box，绝大部分的 default box 都是负样本，导致了正负样本不平衡。在训练过程中，采用了 Hard Negative Mining 的策略（根据confidence loss对所有的box进行排序，**使正负例的比例保持在1:3**） 来平衡正负样本的比率。这样做能提高4%左右。

**参考文献：**<http://www.360doc.com/content/17/0810/10/10408243_678091430.shtml>

<http://blog.csdn.net/u010167269/article/details/52563573>

# 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, k=3,p=1,s=1**  **（pool5前后feature map大小不变）** | (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, k=3,p=1,s=1 |
| **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）** |

**备注**：

1、VOC数据采用6层分类回归，没有最后的c**onv10\_2\_mbox\_loc、conv10\_2\_mbox\_conf**

**层。**如果想将7层分类回归改成6层，需要将多余的层删除（比如onv10\_2\_mbox\_loc

conv10\_2\_mbox\_conf还有conv10等），不然预测时会出错。

2、以feature map上**每个点的中点**为中心（offset=0.5），生成一系列同心的prior box（然

后中心点的坐标会乘以**step**，相当于从feature map位置映射回原图位置）。

＃输入图像的最小尺寸  min\_dim = 300   #######维度

# conv4\_3 ==> 38 x 38   **# step=8，300/8=37.5**

# fc7 ==> 19 x 19   # **step=16，**300/16 =18.75

# conv6\_2 ==> 10 x 10   # **step=32，**300/32 = 9.375

# conv7\_2 ==> 5 x 5   # **step=64，**300/64= 4.6875

# conv8\_2 ==> 3 x 3   # **step=128，**300/128 = 2.34375

# conv9\_2 ==> 1 x 1   # **step=256，**300/256 = 1.171875

3、**flip = True**  ###如果为true，则会翻转每个宽高比。例如，如果有纵横比“r”，我们也

会产生纵横比“1.0 / r”。故产生{1，2，3，1/2，1/3}。

**clip = False** ###做clip操作是为了让prior的候选坐标位置保持在[0,1]范围内。在

caffe.proto文件中有关于参数clip的解释，为”如果为true，则将先验框

裁剪为[0，1]“。

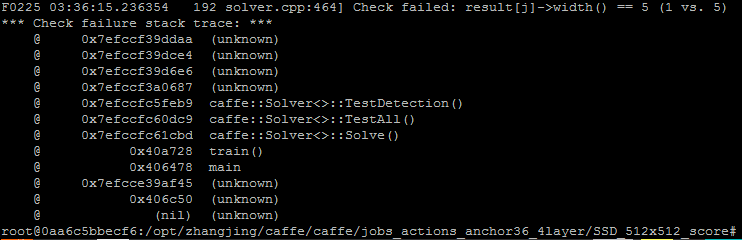
#以上两个参数所产生的结果均在prior\_box\_layer.cpp中实现。

4、基础网络为VGG16。7个分类回归层，每层anchor的个数分布为4、6、6、6、6、

4、4.（共36个）

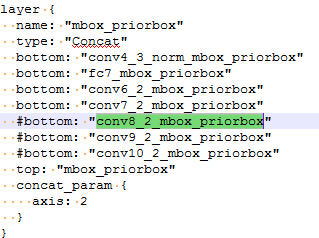
# Error

## 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