单目标检测：足球检测

说明：本文件为markdown文件，可用Typora软件打开

课程演示环境：Ubuntu16.04; cuda 10.1; cudnn7.6.5; Python3.7;

1. 安装YOLOV4

官网： https://github.com/AlexeyAB/darknet

1）克隆AlexyAB/darknet

```
git clone https://github.com/AlexeyAB/darknet
```

2）编译项目

```
cd darknet
```

```
make
```

3）下载预训练权重文件

yolov4.weights

百度网盘下载链接:

链接：https://pan.baidu.com/s/1fBw1QMWmqrzPKI64X1BeMA 提取码：no7k

4）安装测试

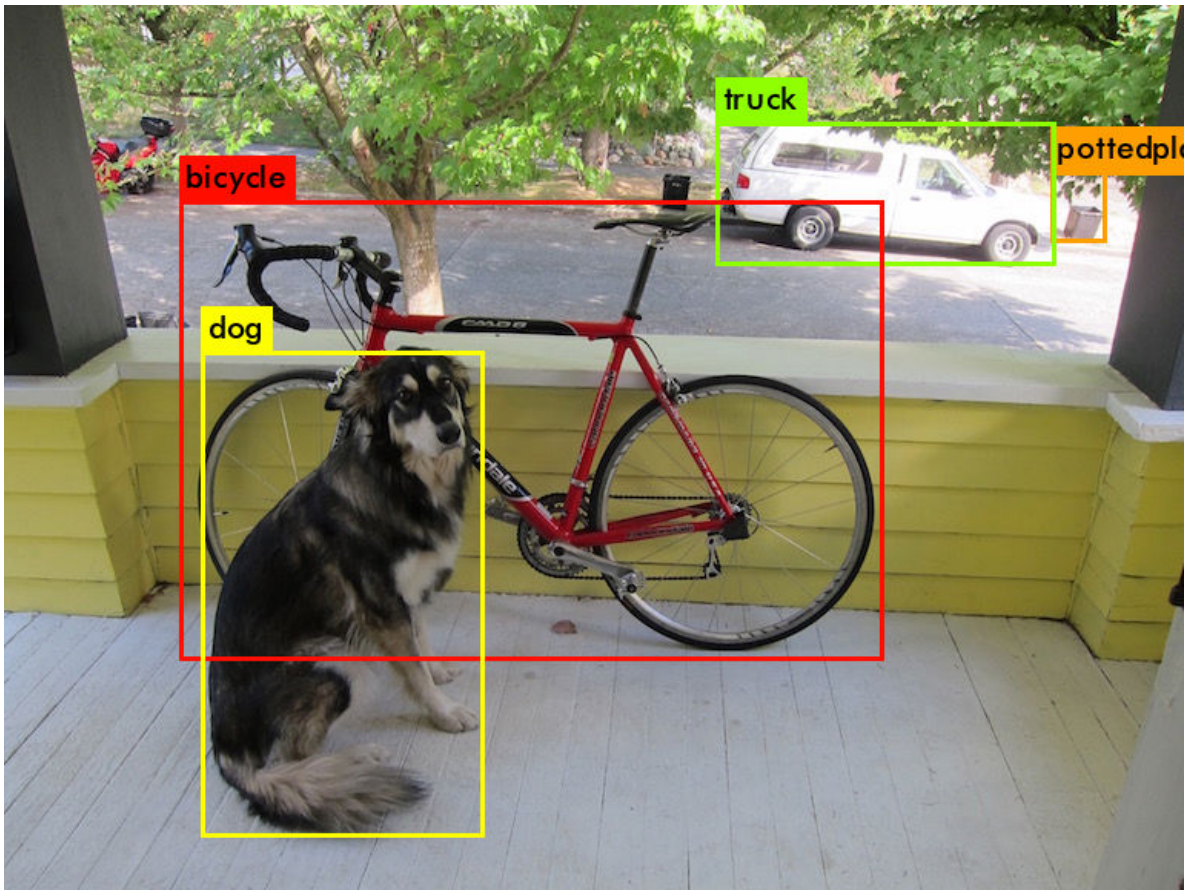
```
./darknet detector test ./cfg/coco.data ./cfg/yolov4.cfg ./yolov4.weights
data/dog.jpg
```

Not compiled with OpenCV, saving to predictions.png instead

```
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, cls_norm: 1.00, scale_x_y: 1.10
nms_kind: greedynms (1), beta = 0.600000
 151 route  147                                           ->   38 x  38 x 256
 152 conv    512       3 x 3/ 2     38 x  38 x 256 ->   19 x  19 x 512 0.852 BF
 153 route  152 116                                     ->   19 x  19 x1024
 154 conv    512       1 x 1/ 1     19 x  19 x1024 ->   19 x  19 x 512 0.379 BF
 155 conv   1024       3 x 3/ 1     19 x  19 x 512 ->   19 x  19 x1024 3.407 BF
 156 conv    512       1 x 1/ 1     19 x  19 x1024 ->   19 x  19 x 512 0.379 BF
 157 conv   1024       3 x 3/ 1     19 x  19 x 512 ->   19 x  19 x1024 3.407 BF
 158 conv    512       1 x 1/ 1     19 x  19 x1024 ->   19 x  19 x 512 0.379 BF
 159 conv   1024       3 x 3/ 1     19 x  19 x 512 ->   19 x  19 x1024 3.407 BF
 160 conv    255       1 x 1/ 1     19 x  19 x1024 ->   19 x  19 x 255 0.189 BF
 161 yolo
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, cls_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
Total BFLOPS 128.459
avg_outputs = 1068395
Loading weights from ./yolov4.weights...
 seen 64, trained: 32032 K-images (500 Kilo-batches_64)
Done! Loaded 162 layers from weights-file
data/dog.jpg: Predicted in 22170.274000 milli-seconds.
bicycle: 92%
dog: 98%
truck: 92%
pottedplant: 33%
Not compiled with OpenCV, saving to predictions.png instead
bai@bai-MS-7A94:~/darknet$
```

测试结果如下：

目录darknet下的predictions.jpg是产生的预测结果图像文件



因只使用CPU，预测时间较长 (20多秒)

5）使用CUDA和OpenCV编译

注意：yolov4项目需要CUDA10.0以上，cuDNN也需要安装对应的版本

OpenCV安装

注意：OpenCV版本<=4.0（不要用最新版本）

本人安装的是opencv 3.4.4。首先到opencv官网下载opencv-3.4.4.tar.gz。执行以下命令

```
tar xvf  opencv-3.4.4.tar.gz
```

```
cd opencv-3.4.4/
```

```
cmake .
```

```
make
```

```
sudo make install
```

在执行上述的cmake时可根据自己的电脑配置和安装的opencv版本情况设置命令参数：

（注意cuda的版本设置）

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D INSTALL_PYTHON_EXAMPLES=ON \
    -D INSTALL_C_EXAMPLES=OFF \
    -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.4.4/modules \
    -D PYTHON_EXCUTABLE=/usr/bin/python \
    -D WITH_CUDA=ON \     # 使用CUDA
    -D WITH_CUBLAS=ON \
    -D DCUDA_NVCC_FLAGS="-D_FORCE_INLINES" \
    -D CUDA_ARCH_BIN="10.1" \     # 需要去官网确认使用的GPU所对应的版本
    -D CUDA_ARCH_PTX="" \
    -D CUDA_FAST_MATH=ON \
    -D WITH_TBB=ON \
    -D WITH_V4L=ON \
    -D WITH_QT=ON \     # 如果qt未安装可以删去此行;若因为未正确安装qt导致的Qt5Gui报错,
    -D WITH_GTK=ON \
    -D WITH_OPENGL=ON \
    -D BUILD_EXAMPLES=ON ..
```

本人使用的cmake命令如下：

```
cmake -D CMAKE_BUILD_TYPE=RELEASE  -D CMAKE_INSTALL_PREFIX=/usr/local  -D
INSTALL_PYTHON_EXAMPLES=ON   -D INSTALL_C_EXAMPLES=OFF    -D
OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.4.4/modules  -D
CUDA_GENERATION=Auto  -D PYTHON_EXCUTABLE=/usr/bin/python    -D WITH_TBB=ON -D
WITH_V4L=ON   -D WITH_GTK=ON  -D WITH_OPENGL=ON   -D BUILD_EXAMPLES=ON ..
```

`sudo make install` 执行完毕后OpenCV编译过程就结束了，接下来就需要配置一些OpenCV的编译环境首先将OpenCV的库添加到路径，从而可以让系统找到。

```
sudo gedit /etc/ld.so.conf.d/opencv.conf
```

执行此命令后打开的可能是一个空白的文件，不用管，只需要在文件末尾添加 /usr/local/lib

执行如下命令使得刚才的配置路径生效

```
sudo ldconfig
```

配置bash

```
sudo gedit /etc/bash.bashrc
```

在最末尾添加 PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig
export PKG_CONFIG_PATH

保存，执行如下命令使得配置生效

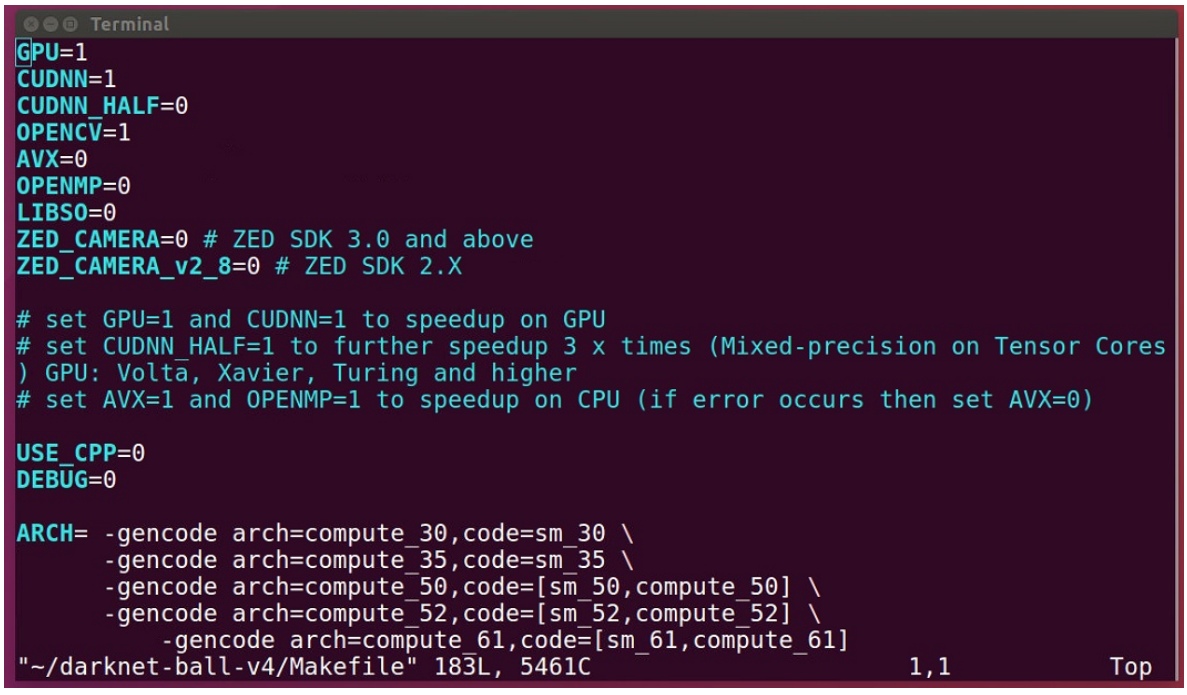
```
source /etc/bash.bashrc
```

更新

```
sudo updatedb
```

修改darknet目录下的Makefile文件

```
GPU=1
CUDNN=1
OPENCV=1
```



然后执行编译命令

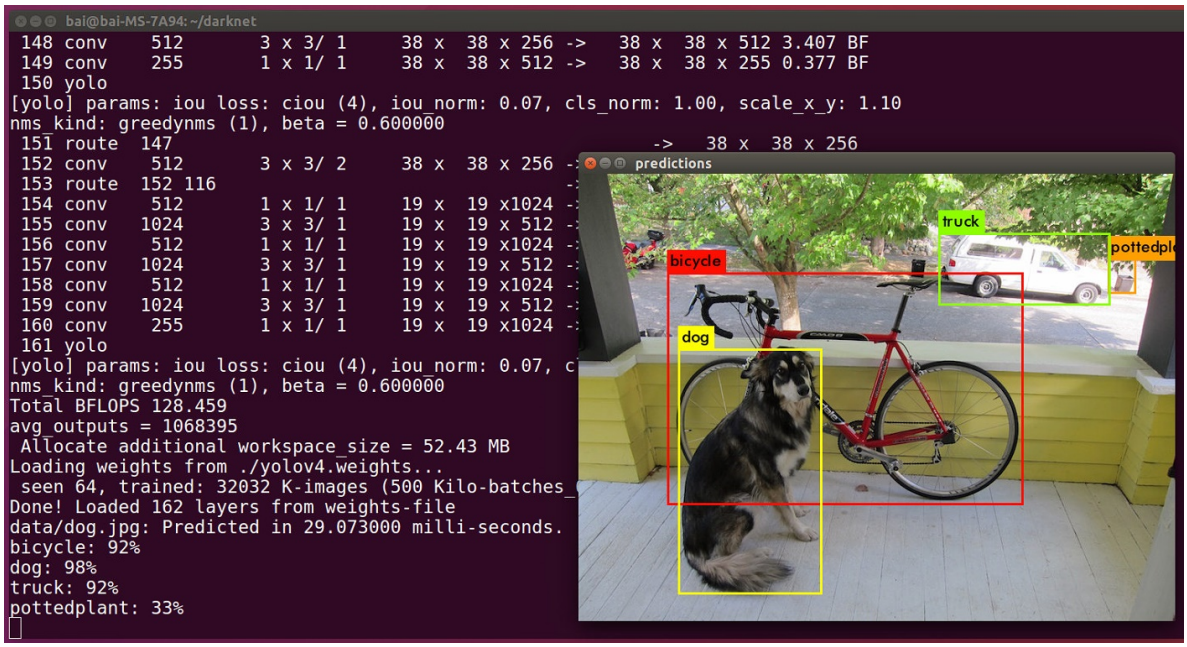
```
make clean
```

```
make
```

6) 测试GPU版本的YOLOv4

测试图片

```
./darknet detector test ./cfg/coco.data ./cfg/yolov4.cfg ./yolov4.weights
data/dog.jpg
```



预测时间减少到约29毫秒

测试opencv

./darknet imtest data/eagle.jpg

2. 标注自己的数据集

1) 安装图像标注工具labelImg

Ubuntu Linux下的安装:

建议使用Python 2.7和Qt4安装。Python 3和Qt5安装容易出问题。

克隆labelImg

git clone https://github.com/tzutalin/labelImg

```
git clone https://github.com/tzutalin/labelImg.git
```

Python 2 + Qt4安装

```
cd ~/labelImg
sudo apt-get install pyqt4-dev-tools
sudo pip install lxml
make qt4py2
python labelImg.py
```

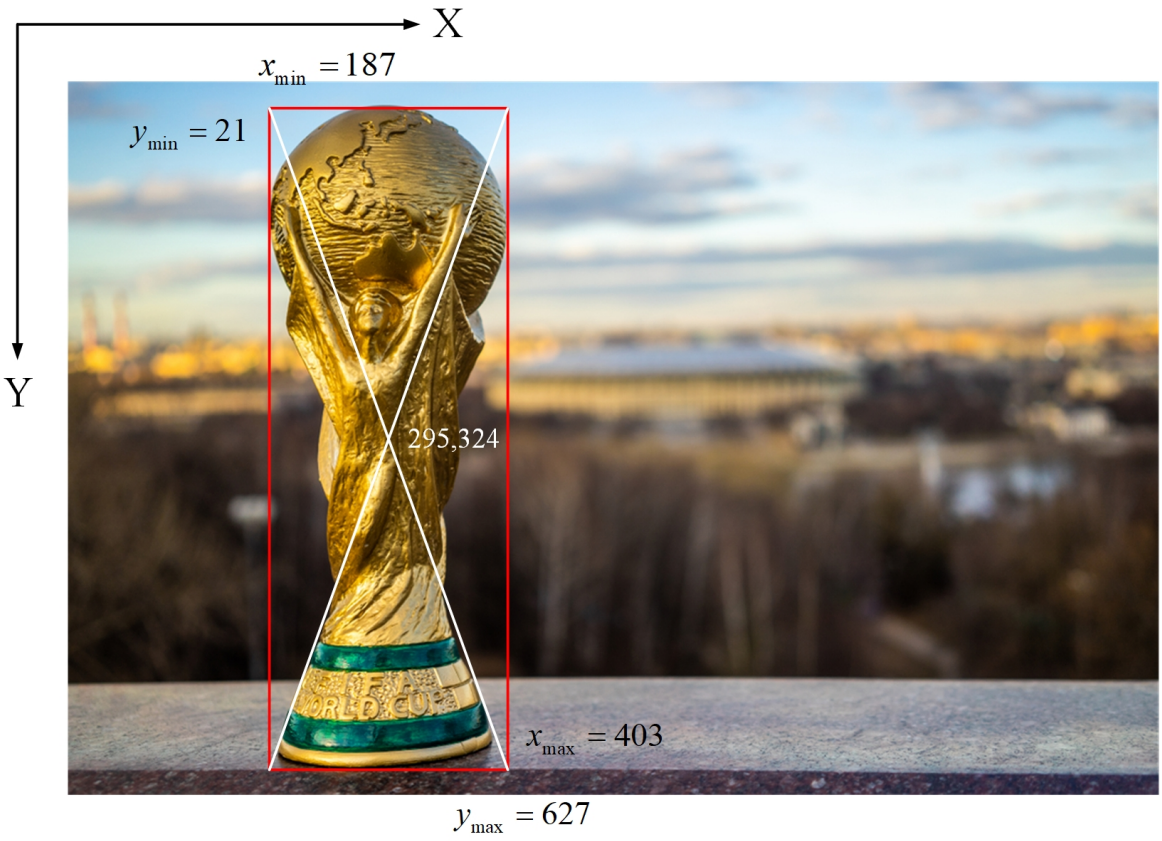
2) 添加自定义类别

修改文件labelImg/data/predefined_classes.txt

```
ball
messi
trophy
```

3）使用labelImg进行图像标注

用labelImg标注生成PASCAL VOC格式的xml标记文件。例如：



width =1000

height = 654

PASCAL VOC标记文件如下：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
- <annotation>
    <folder>bai</folder>
    <filename>trophy.jpg</filename>
    <path>/home/bai/trophy.jpg</path>
  - <source>
      <database>Unknown</database>
    </source>
  - <size>
      <width>1000</width>
      <height>654</height>
      <depth>3</depth>
    </size>
    <segmented>0</segmented>
  - <object>
      <name>trophy</name>
      <pose>Unspecified</pose>
      <truncated>0</truncated>
      <difficult>0</difficult>
    - <bndbox>
        <xmin>187</xmin>
        <ymin>21</ymin>
        <xmax>403</xmax>
        <ymax>627</ymax>
      </bndbox>
    </object>
</annotation>
```

也可以直接生成YOLO格式的txt标记文件如下：

class_id x y w h

```
2 0.295000 0.495413 0.216000 0.926606
```

x = x_center/width = 295/1000 = 0.2950

y = y_center/height = 324/654 = 0.4954

w = (xmax - xmin)/width = 216/1000 = 0.2160

h = (ymax - ymin)/height = 606/654 = 0.9266

class_id: 类别的id编号

x: 目标的中心点x坐标（横向）/图片总宽度

y: 目标的中心的y坐标（纵向）/图片总高度

w: 目标框的宽带/图片总宽度

h: 目标框的高度/图片总高度

可以用python代码实现两种标记格式的转换：

```
def convert(size, box):
    dw = 1./size[0]
    dh = 1./size[1]
    x = (box[0] + box[1])/2.0
    y = (box[2] + box[3])/2.0
    w = box[1] - box[0]
    h = box[3] - box[2]
    x = x*dw
    w = w*dw
    y = y*dh
    h = h*dh
    return (x,y,w,h)
```

box[0]：xmin

box[1]: xmax

box[2]: ymin

box[3]: ymax

3. 准备自己的数据集

1) 下载项目文件：

从百度网盘下载到darknet目录下并解压

- VOCdevkit_ball.tar.gz
- testfiles.tar.gz
- gen_files.py
- reval_voc.py
- voc_eval.py
- draw_pr.py

2）解压建立或自行建立数据集

使用PASCAL VOC数据集的目录结构:

建立文件夹层次为 darknet / VOCdevkit / VOC2007

VOC2007下面建立两个文件夹：Annotations和JPEGImages

JPEGImages放所有的训练和测试图片；Annotations放所有的xml标记文件

3）生成训练集和测试集文件

执行

```
python gen_files.py
```

在VOCdevkit / VOC2007目录下可以看到生成了文件夹labels ， 同时在darknet下生成了两个文件2007_train.txt和2007_test.txt。

2007_train.txt和2007_test.txt分别给出了训练图片文件和测试图片文件的列表，含有每个图片的路径和文件名。

另外，在VOCdevkit / VOC2007/ImageSets/Main目录下生产了两个文件test.txt和train.txt，分别给出了训练图片文件和测试图片文件的列表，但只含有每个图片的文件名（不含路径和扩展名）。

labels下的文件是images文件夹下每一个图像的yolo格式的标注文件，这是由annotations的xml标注文件转换来的。

最终训练只需要：2007_train.txt，2007_test.txt，labels下的标注文件和 VOCdevkit / VOC2007/JPEGImages下的图像文件。

4. 修改配置文件

1）新建data/voc.names文件

可以复制data/voc.names再根据自己情况的修改；可以重新命名如：data/voc-ball.names

2）新建 cfg/voc.data文件

可以复制cfg/voc.data再根据自己情况的修改；可以重新命名如：cfg/voc-ball.data

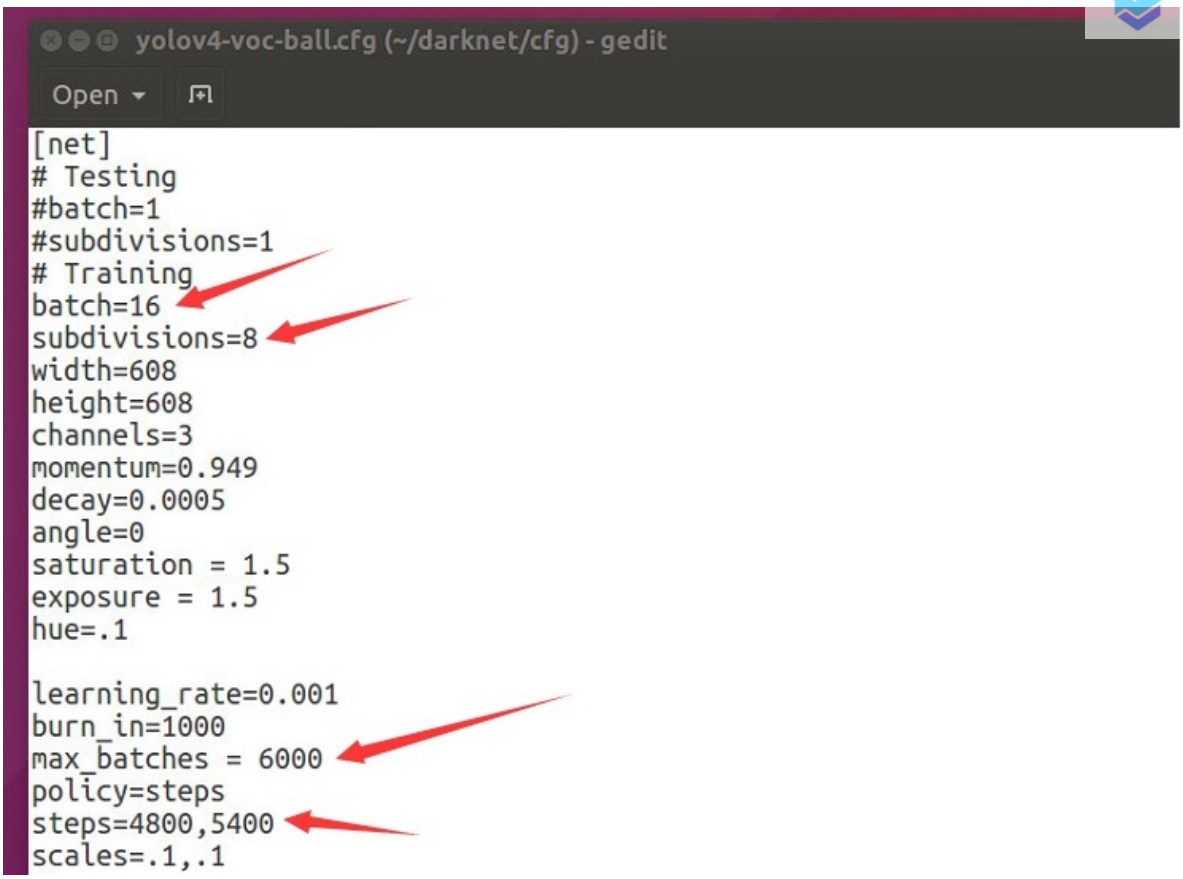
3）新建cfg/yolov4-voc.cfg

可以复制cfg/yolov4-custom.cfg再根据自己情况的修改；可以重新命名cfg/yolov4-voc-ball.cfg：

batch=16 subdivisions=8

max_batches = 6000

steps=4800, 5400



在cfg/yolov4-voc-ball.cfg文件中，三个yolo层和各自前面的convolutional层的参数需要修改：

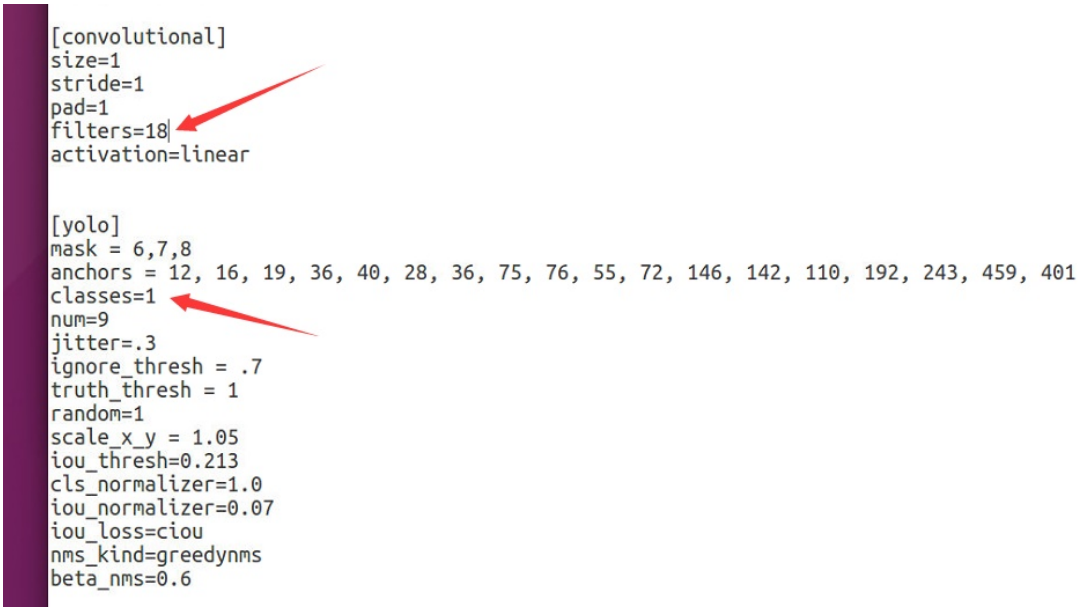
三个yolo层都要改：yolo层中的classes为类别数，每一个yolo层前的convolutional层中的filters =（类别+5）* 3

例如：

yolo层 classes=1， convolutional层 filters=18

yolo层 classes=2， convolutional层 filters=21

yolo层 classes=4，convolutional层 filters=27



5. 训练自己的数据集

1）在`darknet`目录下载预训练权重文件

yolov4.conv.137

这里的训练使用迁移学习，所以下载的yolov4在coco数据集上的预训练权重文件（不含全连接层）

2）训练

```
./darknet detector train cfg/voc-ball.data cfg/yolov4-voc-ball.cfg
yolov4.conv.137
```

如需要显示训练过程的map变化，在命令末尾加-map，即

```
./darknet detector train cfg/voc-ball.data cfg/yolov4-voc-ball.cfg
yolov4.conv.137 -map
```

3) 训练建议

- batch=16
- subdivisions=8
- 把max_batches设置为 (classes*2000)；但最小为4000。例如如果训练3个目标类别，max_batches=6000
- 把steps改为max_batches的80% and 90%；例如steps=4800, 5400。
- 为增加网络分辨率可增大height和width的值，但必须是32的倍数 (height=608, width=608 or 32的整数倍) 。这有助于提高检测精度。

6. 测试训练出的网络模型

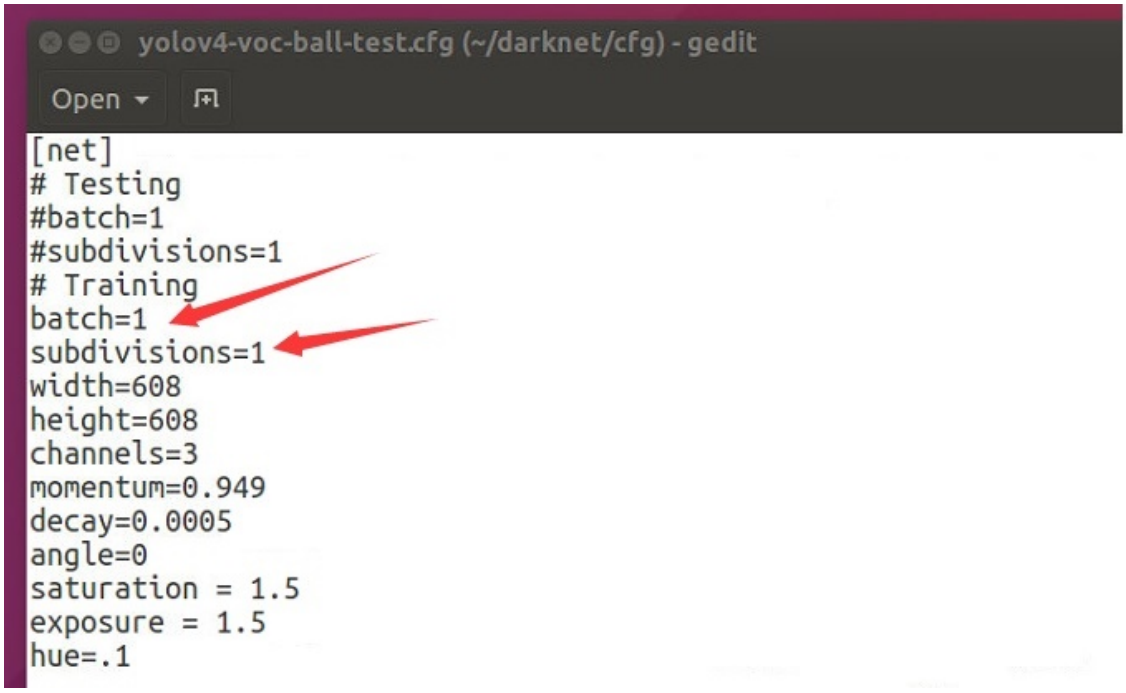
1）创建训练cfg文件

训练好后可以在backup目录下看到权重文件。

尝试test前要修改cfg文件，切换到test模式。可以重新建立一个测试cfg文件, 如yolov4-voc-ball-test.cfg

设置：

batch=1 subdivisions=1



2）测试图片

```
./darknet detector test cfg/voc-ball.data cfg/yolov4-voc-ball-test.cfg
backup/yolov4-voc-ball_final.weights testfiles/img1.jpg
```

3）测试视频

```
./darknet detector demo cfg/voc-ball.data cfg/yolov4-voc-ball-test.cfg
backup/yolov4-voc-ball_final.weights testfiles/messi.mp4
```

7. 性能统计

1) 计算mAP方法1

统计 mAP@IoU=0.50:

```
./darknet detector map cfg/voc-ball.data cfg/yolov4-voc-ball-test.cfg
backup/yolov4-voc-ball_final.weights
```

统计 mAP@IoU=0.75:

```
./darknet detector map cfg/voc-ball.data cfg/yolov4-voc-ball-test.cfg
backup/yolov4-voc-ball_final.weights  -iou_thresh 0.75
```

2) 计算mAP方法2

首先执行

```
./darknet detector valid cfg/voc-ball.data cfg/yolov4-voc-ball-test.cfg
backup/yolov4-voc-ball_final.weights
```

生成results/comp4_det_test_ball.txt文件

然后执行

```
python reval_voc.py  --voc_dir VOCdevkit --year 2007 --image_set test --classes
data/voc-ball.names testball
```

生成testball/ball_pr.pkl文件

画出PR曲线

然后可画出PR曲线，

修改文件draw_pr.py

fr = open('testball/ball_pr.pkl','rb')

执行

```
python draw_pr.py
```

8. Anchor Box先验框聚类分析与修改

1) 使用k-means聚类获得自己数据集的先验框大小

```
./darknet detector calc_anchors cfg/voc-ball.data -num_of_clusters 9 -width 608
-height 608
```

2) 修改cfg文件中的先验框大小

3) 重新训练和测试