

目标检测：足球检测

1. 安装darknet

官网: <https://pjreddie.com/darknet/yolo/>

1) 克隆darknet

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

2) 编译项目

```
cd darknet
```

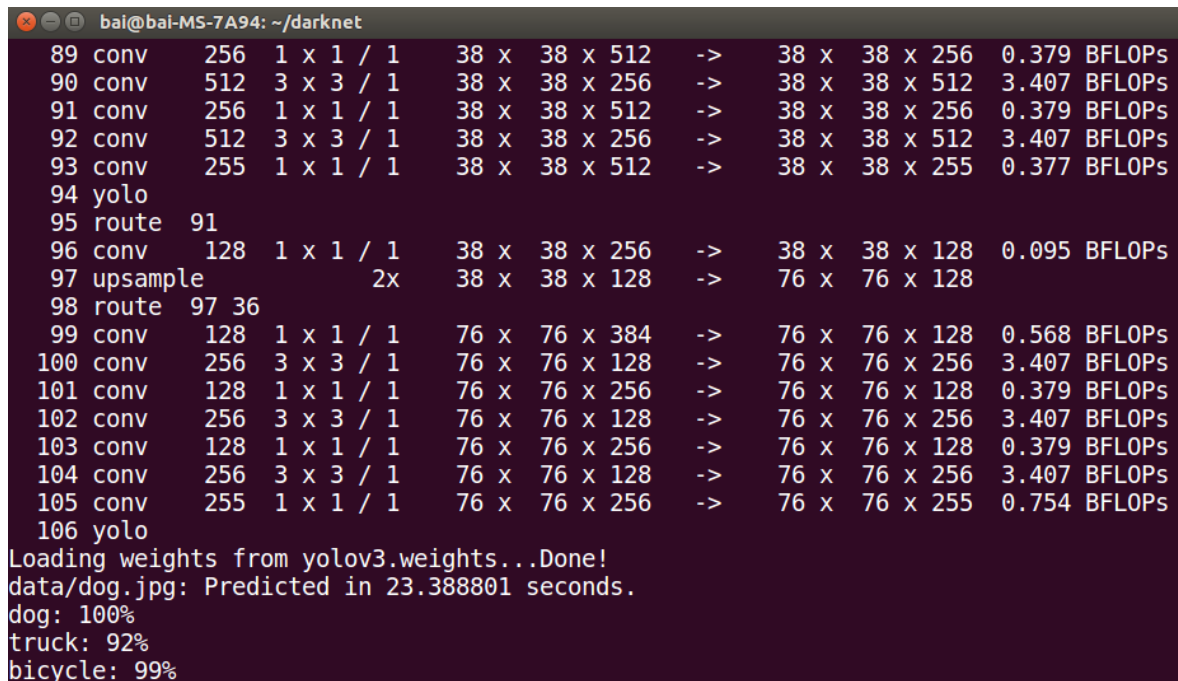
```
make
```

3) 下载预训练权重文件 (download the pre-trained weight file)

```
wget https://pjreddie.com/media/files/yolov3.weights
```

4) 测试

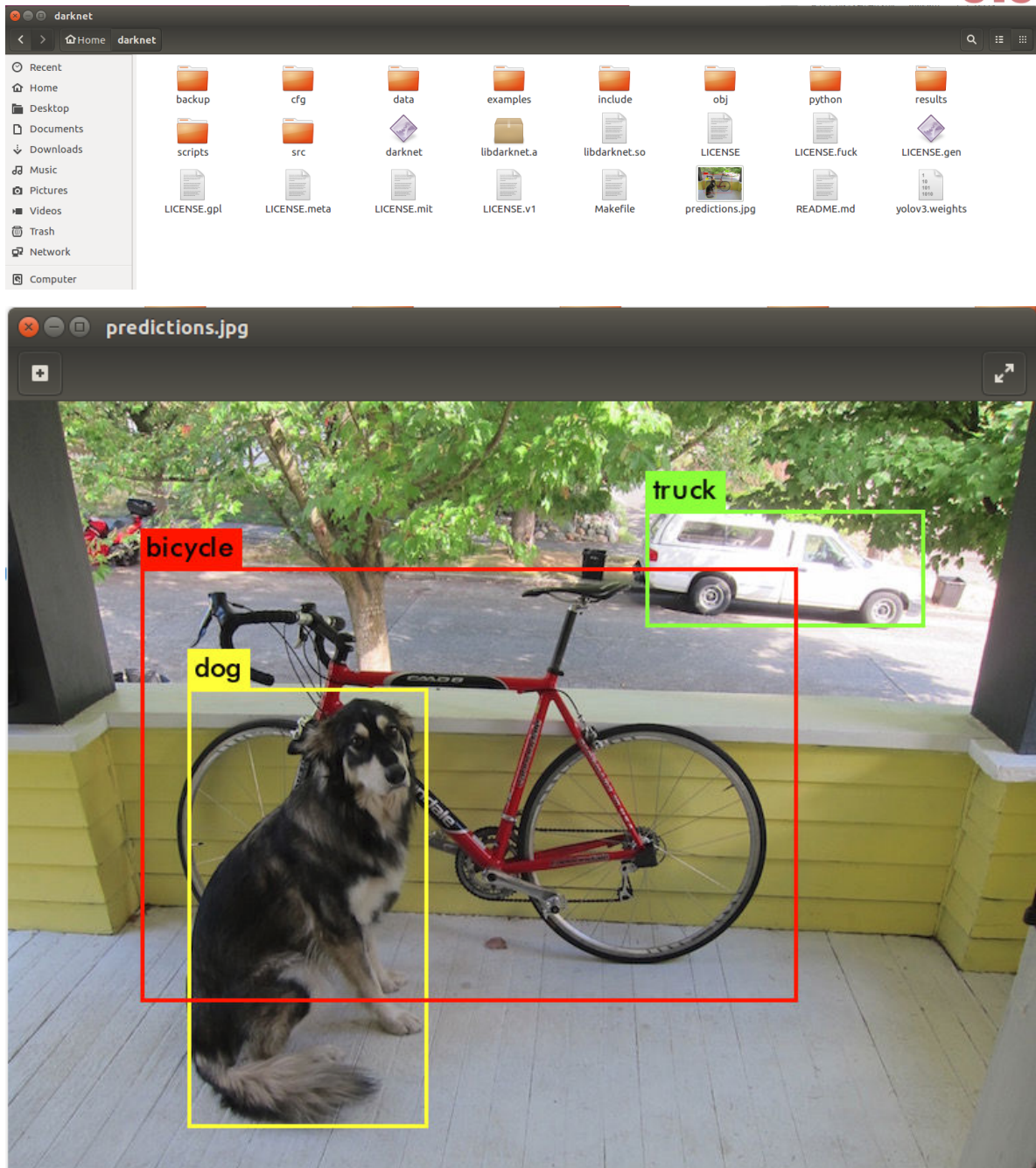
```
./darknet detect cfg/yolov3.cfg yolov3.weights data/dog.jpg
```



```
bai@bai-MS-7A94: ~/darknet
89 conv 256 1 x 1 / 1 38 x 38 x 512 -> 38 x 38 x 256 0.379 BFL0Ps
90 conv 512 3 x 3 / 1 38 x 38 x 256 -> 38 x 38 x 512 3.407 BFL0Ps
91 conv 256 1 x 1 / 1 38 x 38 x 512 -> 38 x 38 x 256 0.379 BFL0Ps
92 conv 512 3 x 3 / 1 38 x 38 x 256 -> 38 x 38 x 512 3.407 BFL0Ps
93 conv 255 1 x 1 / 1 38 x 38 x 512 -> 38 x 38 x 255 0.377 BFL0Ps
94 yolo
95 route 91
96 conv 128 1 x 1 / 1 38 x 38 x 256 -> 38 x 38 x 128 0.095 BFL0Ps
97 upsample 2x 38 x 38 x 128 -> 76 x 76 x 128
98 route 97 36
99 conv 128 1 x 1 / 1 76 x 76 x 384 -> 76 x 76 x 128 0.568 BFL0Ps
100 conv 256 3 x 3 / 1 76 x 76 x 128 -> 76 x 76 x 256 3.407 BFL0Ps
101 conv 128 1 x 1 / 1 76 x 76 x 256 -> 76 x 76 x 128 0.379 BFL0Ps
102 conv 256 3 x 3 / 1 76 x 76 x 128 -> 76 x 76 x 256 3.407 BFL0Ps
103 conv 128 1 x 1 / 1 76 x 76 x 256 -> 76 x 76 x 128 0.379 BFL0Ps
104 conv 256 3 x 3 / 1 76 x 76 x 128 -> 76 x 76 x 256 3.407 BFL0Ps
105 conv 255 1 x 1 / 1 76 x 76 x 256 -> 76 x 76 x 255 0.754 BFL0Ps
106 yolo
Loading weights from yolov3.weights...Done!
data/dog.jpg: Predicted in 23.388801 seconds.
dog: 100%
truck: 92%
bicycle: 99%
```

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，超过23秒

5) 使用CUDA和OPENCV编译 (Compiling With CUDA and OPENCV)

- CUDA安装 (见安装CUDA文件)
- OpenCV安装

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

```
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_EXECUTABLE=/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="9.0" \      # 这个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 \      # 计算速度更快但是相对不精确
-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_EXECUTABLE=/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

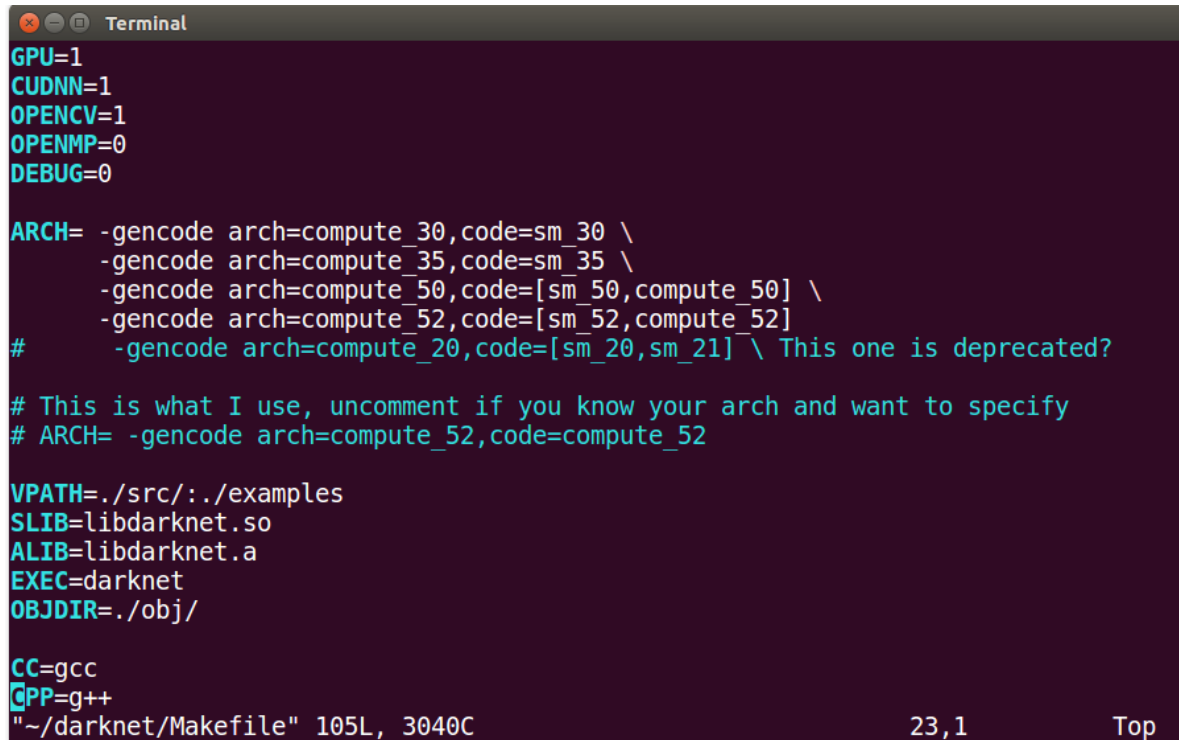
```
source /etc/bash.bashrc
```

更新

```
sudo updatedb
```

- 修改darknet的Makefile

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

A terminal window titled "Terminal" with a dark background and light-colored text. It displays the configuration for the darknet Makefile. The variables are set as follows: GPU=1, CUDNN=1, OPENCV=1, OPENMP=0, and DEBUG=0. The ARCH variable is configured with multiple gencode options for different GPU architectures: compute_30, compute_35, compute_50, compute_52, and compute_20. A comment indicates that compute_20 is deprecated. The VPATH is set to ./src/./examples, SLIB to libdarknet.so, ALIB to libdarknet.a, EXEC to darknet, and OBJDIR to ./obj/. The compiler is set to gcc and the linker to g++. The bottom of the terminal shows the file path ~/darknet/Makefile and some statistics (105L, 3040C) and a cursor position (23,1) with a "Top" button.

```
GPU=1  
CUDNN=1  
OPENCV=1  
OPENMP=0  
DEBUG=0  
  
ARCH= -gencode arch=compute_30,code=sm_30 \  
      -gencode arch=compute_35,code=sm_35 \  
      -gencode arch=compute_50,code=[sm_50,compute_50] \  
      -gencode arch=compute_52,code=[sm_52,compute_52]  
#      -gencode arch=compute_20,code=[sm_20,sm_21] \ This one is deprecated?  
  
# This is what I use, uncomment if you know your arch and want to specify  
# ARCH= -gencode arch=compute_52,code=compute_52  
  
VPATH=./src/./examples  
SLIB=libdarknet.so  
ALIB=libdarknet.a  
EXEC=darknet  
OBJDIR=./obj/  
  
CC=gcc  
CXX=g++  
"~/darknet/Makefile" 105L, 3040C 23,1 Top
```

然后执行

```
make clean
```

```
make
```

6) 测试GPU版本的yolo

```
./darknet detect cfg/yolov3.cfg yolov3.weights data/dog.jpg
```

```

bai@bai-MS-7A94: ~/darknet
89 conv 256 1 x 1 / 1 38 x 38 x 512 -> 38 x 38 x 256 0.379 BFLOPs
90 conv 512 3 x 3 / 1 38 x 38 x 256 -> 38 x 38 x 512 3.407 BFLOPs
91 conv 256 1 x 1 / 1 38 x 38 x 512 -> 38 x 38 x 256 0.379 BFLOPs
92 conv 512 3 x 3 / 1 38 x 38 x 256 -> 38 x 38 x 512 3.407 BFLOPs
93 conv 255 1 x 1 / 1 38 x 38 x 512 -> 38 x 38 x 255 0.377 BFLOPs
94 yolo
95 route 91
96 conv 128 1 x 1 / 1 38 x 38 x 256 -> 38 x 38 x 128 0.095 BFLOPs
97 upsample 2x 38 x 38 x 128 -> 76 x 76 x 128
98 route 97 36
99 conv 128 1 x 1 / 1 76 x 76 x 384 -> 76 x 76 x 128 0.568 BFLOPs
100 conv 256 3 x 3 / 1 76 x 76 x 128 -> 76 x 76 x 256 3.407 BFLOPs
101 conv 128 1 x 1 / 1 76 x 76 x 256 -> 76 x 76 x 128 0.379 BFLOPs
102 conv 256 3 x 3 / 1 76 x 76 x 128 -> 76 x 76 x 256 3.407 BFLOPs
103 conv 128 1 x 1 / 1 76 x 76 x 256 -> 76 x 76 x 128 0.379 BFLOPs
104 conv 256 3 x 3 / 1 76 x 76 x 128 -> 76 x 76 x 256 3.407 BFLOPs
105 conv 255 1 x 1 / 1 76 x 76 x 256 -> 76 x 76 x 255 0.754 BFLOPs
106 yolo
Loading weights from yolov3.weights...Done!
data/dog.jpg: Predicted in 0.035271 seconds.
dog: 100%
truck: 92%
bicycle: 99%

```

预测时间减少到约0.035秒

```
./darknet imtest data/eagle.jpg
```

2. 给自己的数据集打标签

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
```

Python 2 + Qt4安装

```

cd ~/labelImg
sudo apt-get install pyqt4-dev-tools
sudo pip install lxml
make qt4py2
python labelImg.py
python labelImg.py [IMAGE_PATH] [PRE-DEFINED CLASS FILE]

```

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;



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. 整理自己的数据集

1) 下载项目文件:

从百度网盘下载到darknet目录下并解压

链接: <https://pan.baidu.com/s/1R-azYMOEwOZ5dQpfi2OMVQ>
提取码: guk3

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

2) 可选: 建立目录格式

使用PASCAL VOC目录格式:

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) 生成训练和测试文件

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目录下生成了两个文件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JPEGImages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/yolov3-voc.cfg

可以复制cfg/yolov3-voc.cfg再根据自己情况的修改; 可以重新命名cfg/yolov3-voc-ball.cfg:

在cfg/yolov3-voc.cfg文件中, 三个yolo层和各自前面的conv层的参数需要修改:

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

例如:

yolo层 classes=1, conv层 filters=18

yolo层 classes=2, conv层 filters=21

yolo层 classes=4, conv层 filters=27

5. 训练自己的数据集

1) 在 darknet 目录下载权重文件:

wget <https://pjreddie.com/media/files/darknet53.conv.74>

这里的训练使用迁移学习, 所以下载的yolo预训练的权重文件(不含全连接层)

2) 训练

```
./darknet detector train cfg/voc-ball.data cfg/yolov3-voc-ball.cfg  
darknet53.conv.74
```

如需要存储训练日志, 执行

```
./darknet detector train cfg/voc-ball.data cfg/yolov3-voc-ball.cfg
darknet53.conv.74 2>1 | tee visualization/train_yolov3_ball.log
```

执行前应建立visualization目录。可通过将visualization.zip解压到darknet项目目录下。

训练时的输出信息： Region 106 Avg IOU: 0.794182, Class: 0.999382, Obj: 0.966953, No Obj: 0.000240, .5R: 1.000000, .75R: 0.750000, count: 4

- Region 106：网络层的索引为106
- Region Avg IOU: 0.794182：表示在当前l.batch (batch != subdivs) 内的图片的平均IOU
- Class: 0.999382：标注目标分类的正确率，期望该值趋近于1。
- Obj: 0.966953：检测目标的平均目标置信度，越接近1越好。
- No Obj: 0.000793：检测目标的平均目标性得分。
- .5R: 1.0：模型检测出的正样本(iou>0.5)与实际的正样本的比值。
- .75R: 0.75 模型检测出的正样本(iou>0.75)与实际的正样本的比值。
- count: 4：count后的值是当前l.batch (batch != subdivs) 内图片中包含正样本的图片的数量。

上述输出信息是从yolo_layer.c中的函数void forward_yolo_layer(const layer l, network net)打印输出的：

```
printf("Region %d Avg IOU: %f, Class: %f, Obj: %f, No Obj: %f, .5R: %f, .75R: %f, count: %d\n", net.index, avg_iou/count, avg_cat/class_count, avg_obj/count, avg_anyobj/(l.w*l.h*l.n*l.batch), recall/count, recall75/count, count);
```

3) 训练log文件分析

在visualization文件夹下，执行：

```
cd visualization
```

```
python extract_log.py
```

得到两个文件: train_log_loss.txt, train_log_iou.txt

改变其中的lines的值

然后，执行：

```
python train_loss_visualization.py
```

```
python train_iou_visualization.py
```

得到avg_loss.png和Region Avg IOU.png

4) 训练建议

- batch=64
- subdivisions=16 (显存大时可用8)
- 把max_batches设置为 (classes*2000); 但最少为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 .cfg-file (height=608, width=608 or any value multiple of 32) 。这有助于提高检测精度。

因此训练时使用

- batch=64
- subdivisions=16
- width=608
- height=608
- max_batches = 4000
- policy=steps
- steps=3200,3600

6. 测试训练出的网络模型

训练好后可以在[backup](#)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yolov3-voc-ball-test.cfg

测试图片:

```
./darknet detector test cfg/voc-ball.data cfg/yolov3-voc-ball-test.cfg
backup/yolov3-voc-ball_final.weights testfiles/img1.jpg
```

测试视频:

```
./darknet detector demo cfg/voc-ball.data cfg/yolov3-voc-ball-test.cfg
backup/yolov3-voc-ball_final.weights testfiles/messi.mp4
```

7. 性能统计

计算mAP

首先执行

```
./darknet detector valid cfg/voc-ball.data cfg/yolov3-voc-ball-test.cfg
backup/yolov3-voc-ball_final.weights
```

生成results/comp4_det_test_ball.txt文件

然后执行

```
python eval_voc.py --voc_dir /home/bai/darknet/VOCdevkit --year 2007 --
image_set test --classes /home/bai/darknet/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. 先验框聚类与修改

1) 使用k-means聚类获得自己数据集的先验框大小

修改gen_anchors.py文件

```
width_in_cfg_file = 608.  
height_in_cfg_file = 608.
```

执行

```
python gen_anchors.py
```

得到的anchor大小

anchorbox.w*32

anchorbox.h*32

2) 修改cfg文件中的先验框大小

3) 重新训练和测试