# 在服务器上用YOLOv4训练和测试数据集

参考链接：

YOLO V4的paper: <https://arxiv.org/abs/2004.10934>

YOLO V4的代码地址: <https://github.com/AlexeyAB/darknet>

<https://blog.csdn.net/weixin_42770591/article/details/105863701>

<https://blog.csdn.net/Creama_/article/details/106209388>

# 一、编译darknet

（1）下载代码

git clone <https://github.com/AlexeyAB/darknet.git>

（2）在darknet下找到Makefile文件修改如下：

GPU=1 #打开GPU训练和测试

CUDNN=1 #采用Cudnn训练和测试

CUDNN\_HALF=0

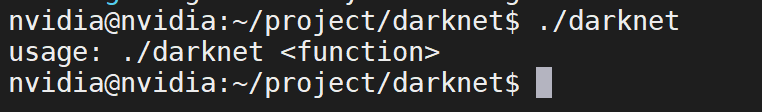
OPENCV=1 #采用OpenCV进行训练

AVX=0

OPENMP=0

LIBSO=1 #启用python接口

（3）然后命令行输入make -j32进行编译，编译完成后输入./darknet，如果出现usage: ./darknet <fuction> 说明编译成功。



（4）采用作者训练好的权值文件yolov4.weights测试一下，下载后放在darknet目录下，在命令行输入：./darknet detect cfg/yolov4.cfg yolov4.weights data/horses.jpg，正常会检测出目标位置和类别

# 二、训练PASCAL VOC2007数据集

作者在github上给的是训练PASCAL VOC2007+2012案例。

（1）准备预训练模型和数据集

首先要下载预训练模型yolov4.conv.137 (https://drive.google.com/file/d/1JKF-bdIklxOOVy-2Cr5qdvjgGpmGfcbp/view)，并放在darknet目录下，将VOC2007数据集放在任意目录下，这里我选择darknet/data/voc/VOCdevkit目录。

（2）生成darknet需要的label文件

将darknet/scripts目录下的voc\_label.py文件复制到darknet/data/voc目录下：cp scripts/voc\_label.py data/voc/，并修改voc\_label.py文件。

执行python voc\_label.py

运行结果产生五个文件2007\_train.txt, 2007\_test.txt, 2007\_val.txt, 2012\_train.txt, 2012\_test.txt, 2012\_val.txt, train.all.txt, train.txt

（3）修改几个配置文件

1.cfg/voc.data

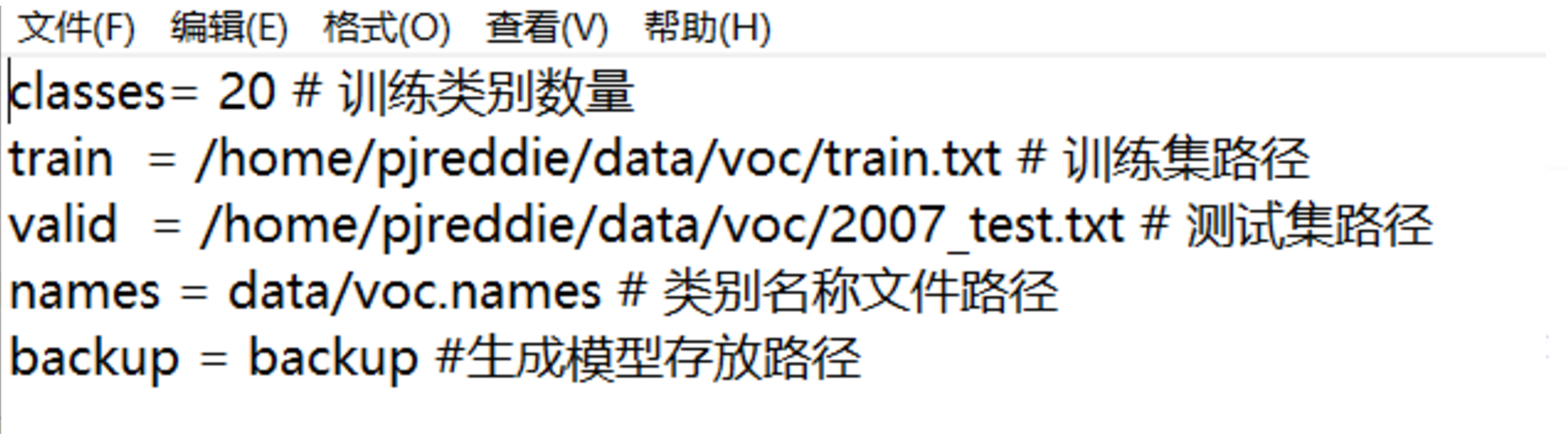
classes= 20

train = /home/zmh/darknet/data/voc/train.txt # 修改为训练集路径

valid = /home/zmh/darknet/data/voc/2007\_test.txt # 修改为测试集路径

names = data/voc.names # 类别名称文件

backup = backup # 模型保存文件夹



2.cfg/yolov4-obj.cfg

将cfg/yolov4-custom.cfg复制，并将复制后的文件命名为yolov4-obj.cfg：

cd cfg

cp yolov4-custom.cfg yolov4-obj.cfg

修改yolov4-obj.cfg:

**第1处**

[net]

#Testing

#batch=1

#subdivisions=1

#Training

batch=64

subdivisions=16 # 如果训练时出现out of memory，可以修改为32或者64

width=608 # 修改为416

height=608 # 修改为416

channels=3

**第2处**

learning\_rate=0.001

burn\_in=1000

max\_batches = 500500 # 修改为(不小于类别数\*2000)

policy=steps

steps=400000,450000 # 修改为0.8\*max\_batches，0.9\*max\_batches

scales=.1,.1

**第3、4、5处**

ctrl+f查找yolo，一共有三处，修改三处yolo上面的filters，以及yolo下面的classes：

[convolutional]

size=1

stride=1

pad=1

filters=255 # 修改为(类别数+5)\*3

activation=linear

[yolo]

mask = 0,1,2

anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243, 459, 401

#修改为自己数据集的anchor大小

classes=80 # 修改为类别数，20

num=9

# 三、训练

开始训练：

./darknet detector train cfg/voc.data cfg/yolov4-obj.cfg yolov4.conv.137 -map -dont\_show -gpus 0,1 | tee yolo\_train.log

训练得到的权值文件保存在darknet/backup中。

参数解析如下：

不实时显示loss和map图（可以在文件夹下看到）：-dont\_show

添加map计算：-map

使用多GPU训练：-gpus 0,1,2,3

| tee yolo\_train.log：如果要保存log文件：# tee后面是log文件名字，自己设置

# 四、测试

测试图片：

./darknet detector test cfg/voc.data cfg/yolov4-obj.cfg backup/yolov4-obj\_best.weights data/person.jpg

测试视频（需要安装OpenCV）：

./darknet detector demo cfg/voc.data cfg/yolov4-obj.cfg backup/yolov4-obj\_best.weights test.mp4 -out\_filename output.mp4

FPS很小的话，加一个-dont\_show或者最小化xmanger就行了。

# 五、训练自己的数据集（制作特定场景数据集）

完成了VOC2007的训练和测试过程，离训练自己的数据集只有一步之遥了，为了方便处理，我把自己的数据集整理成PASCAL VOC的格式，具体过程见轻松自制PASCAL VOC数据集（https://blog.csdn.net/Creama\_/article/details/105974455）。

假设自己的数据集已经被制作成了PASCAL VOC格式了，并且命名为VOC 2007。如果还没有下载预训练模型yolov4.conv.137，需要先下载，下载完成放在darknet目录下，并将自己的数据集(VOC 2007)放在darknet/data/voc/VOCdevkit目录下。接下来进行以下几个跟训练PASCAL VOC数据集类似的操作：

1.生成darknet需要的label文件

如果之前已经修改过voc\_label.py文件了，重新python voc\_label.py就行了，然后会生成五个新的文件：2007\_train.txt, 2007\_test.txt, 2007\_val.txt, train.all.txt, train.txt。

如果之前没有训练过PASCAL VOC数据集，将darknet/scripts目录下的voc\_label.py文件复制到darknet/data/voc目录下：cp scripts/voc\_label.py data/voc/，并修改voc\_label.py文件。

2.修改voc.data文件

如果训练过PASCAL VOC，只要修改其中的classes。否则根据注释修改：

classes= 2 # 改为自己数据集的类别数

train = /home/rencong/darknet/data/voc/train.txt # 修改为训练集路径

valid = /home/rencong/darknet/data/voc/2007\_test.txt # 修改为测试集路径

names = data/voc.names # 类别名称文件，下面会讲怎么修改里面的内容

backup = backup # 模型保存文件夹

3.修改cfg/yolov4-obj.cfg文件

如果没有训练过PASCAL VOC，需要将cfg/yolov4-custom.cfg复制，并将复制后的文件命名为yolov4-obj.cfg：

cd cfg

cp yolov4-custom.cfg yolov4-obj.cfg

然后修改yolov4-obj.cfg:

**第1处：**

[net]

#Testing

#batch=1

#subdivisions=1

#Training

batch=64

subdivisions=16 # 如果训练时出现out of memory，可以修改为32或者64

width=608 # 修改为416

height=608 # 修改为416

channels=3

下面几处即使训练过PASCAL VOC，同样要对应修改**，第2处**：

learning\_rate=0.001

burn\_in=1000

max\_batches = 4000 # 修改为(类别数\*2000)

policy=steps

steps=3200,3600 # 修改为0.8\*max\_batches，0.9\*max\_batches

scales=.1,.1

**以及第3-5处**：

ctrl+f查找yolo，修改三处yolo上面的filters，以及yolo下面的classes：

[convolutional]

size=1

stride=1

pad=1

filters=21 # 修改为(类别数+5)\*3

activation=linear

[yolo]

mask = 0,1,2

anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243, 459, 401

classes=2 # 修改为类别数，20

num=9

4.修改data/voc.names文件（注意，之前训练PASCAL数据集是没有这个步骤的）

voc.names文件中保存的是VOC数据集的类别（20个），我们要修改为自己数据集的类别。

aeroplane

bicycle

bird

boat

bottle

bus

car

cat

chair

cow

diningtable

dog

horse

motorbike

person

pottedplant

sheep

sofa

train

tvmonitor

改为自己的数据集类别：

green light

red light

5.训练

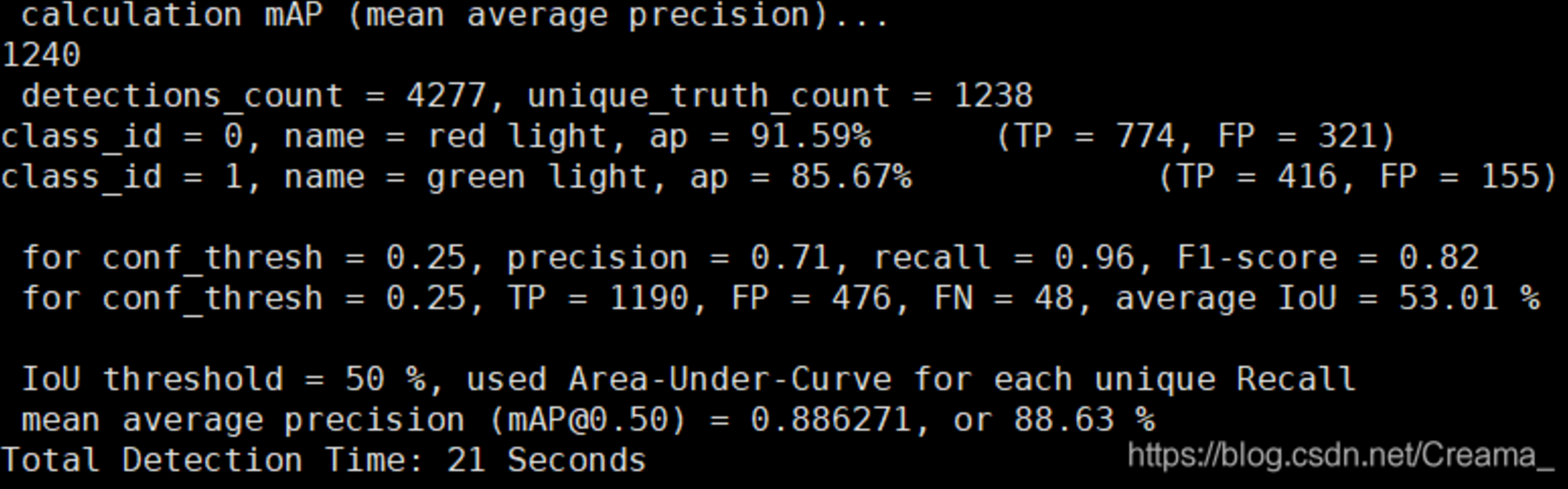
训练之前，转移或删除darknet/backup中之前得到的权值文件。

./darknet detector train cfg/voc.data cfg/yolov4-obj.cfg yolov4.conv.137 -map -dont\_show -gpus 0,1 | tee yolo\_train.log

如果要计算每个权值文件的mAP值，可以：

./darknet detector map cfg/voc.data cfg/yolov4-obj.cfg backup/yolov4-obj\_best.weights

然后可以选一个mAP值最高的权值文件测试。



6.测试

测试单张图片：

./darknet detector test cfg/voc.data cfg/yolov4-obj.cfg backup/yolov4-obj\_best.weights

测试视频(需要OpenCV)：

./darknet detector demo cfg/voc.data cfg/yolov4-obj.cfg backup/yolov4-obj\_best.weights -ext\_output test.mp4

# 六、批量测试图片并保存（可选）

如果要测试大量图片，一张张测试是非常麻烦的，为了能够批量测试图片，并保存在指定文件夹中，需要以下操作：

1.修改darknet/src/detector.c文件

在detector.c文件的开头位置，添加GetFilename函数，为了解决上个版本文件名长度的问题，这里做了修改，直接将上个版本的常数用变量length替代，length是测试图片的文件名，因为要去掉".jpg"，所以strncpy那里要用"length-4"。

2.准备包含所有需要检测图片路径的txt文件

将需要检测的图片放在darknet/data/test文件夹下，然后在darknet/data下新建一个py文件：

touch make\_test\_txt.py

文件中的内容为：

# coding: utf-8

import os

paths = "./test" # 测试图片的路径

f = open('test.txt', 'w') # 最后得到的图片路径txt文件

filenames = os.listdir(paths)

filenames.sort()

for filename in filenames:

out\_path = "data/test/" + filename # 引号内为测试图片文件夹的路径

print(out\_path)

f.write(out\_path + '\n')

f.close()

然后运行make\_test\_txt.py，在data目录下会生成一个test.txt文件。

3.批量测试

在darknet目录下新建一个文件夹output，运行：

./darknet detector test cfg/voc.data cfg/yolov4-obj.cfg backup/ yolov4-obj\_best.weights -ext\_output -dont\_show <data/test.txt> result.txt

注意，保存测试图片文件名的test.txt文件两边有< >，result.txt是把检测的文本输出打印出来，可有可无，检测的图片输出保存在output文件夹中。

# 七、YOLOv4后处理：显示置信度、保存检测框的内容到本地

**（1）显示置信度**

打开src/image.c

查找"draw\_detections\_v3"，找到这个函数的定义部分，注意是定义部分。拉到函数最后一部分，只要在对应位置加上三句：

char buff[20];

sprintf(buff, "(%2.0f%%)", selected\_detections[i].det.prob[selected\_detections[i].best\_class] \* 100);

strcat(labelstr, buff);

即可，然后保存，重新make。

if (alphabet) {

char labelstr[4096] = { 0 };

char buff[20]; //加上这句

strcat(labelstr, names[selected\_detections[i].best\_class]);

sprintf(buff, "(%2.0f%%)", selected\_detections[i].det.prob[selected\_detections[i].best\_class] \* 100); //加上这句

strcat(labelstr, buff); //加上这句

int j;

for (j = 0; j < classes; ++j) {

if (selected\_detections[i].det.prob[j] > thresh && j != selected\_detections[i].best\_class) {

strcat(labelstr, ", ");

strcat(labelstr, names[j]);

}

}

image label = get\_label\_v3(alphabet, labelstr, (im.w\*.01));

draw\_label(im, top + width, left, label, rgb);

free\_image(label);

}

**（2）改变检测框的粗细**

打开src/image.c

查找"draw\_detections\_v3"，找到函数定义部分，在函数体里找到"draw\_box\_width"，其中变量width就是检测框的粗细。作者是这样定义的：

int width = im.h \* .002;

if (width < 1)

width = 1;

变大变小怎么变，你来定。

**（3）保存检测框的内容到本地**

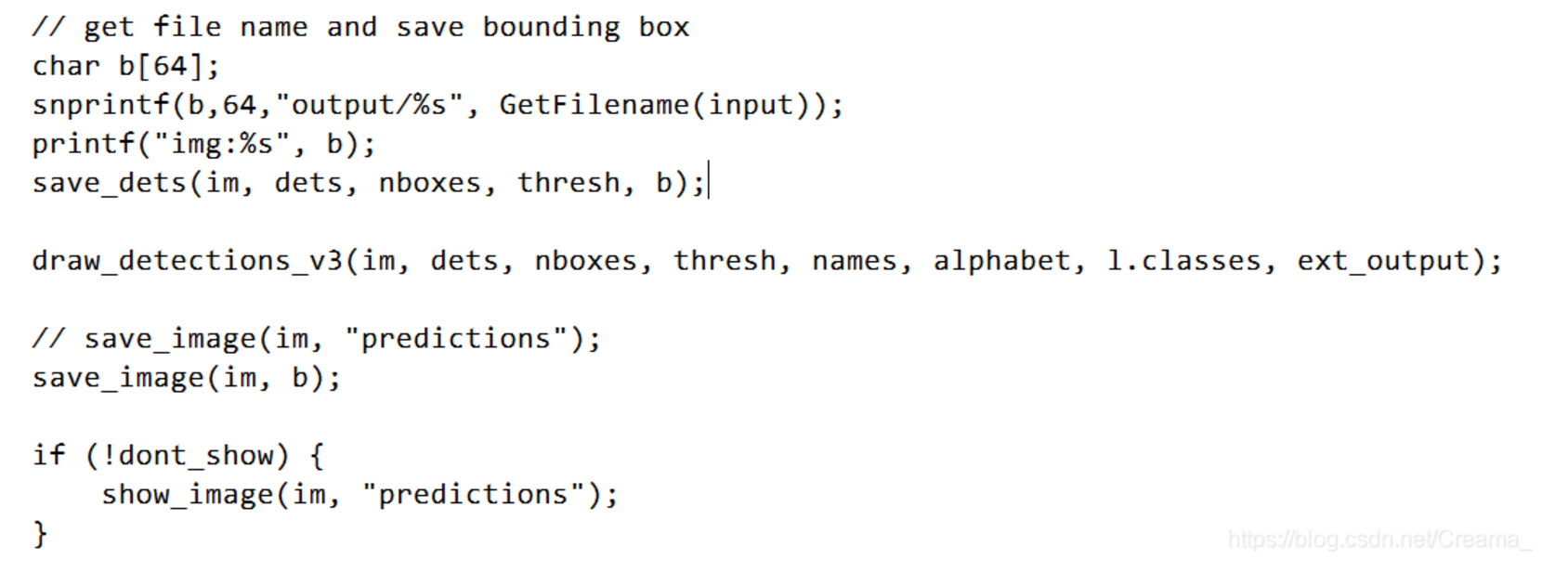
保存检测框的内容到本地

1.修改detector.c

这一部分是在增加了批量检测图片的基础上，将所有检测框中的内容保存到本地。

打开src/detector.c

查找"test\_detector"，找到这个函数的定义部分，再找到里面的draw\_detections\_v3， 如下图所示：



在这里插入图片描述

这里有一部分是之前批量保存图片的代码，为了在画框之前保存检测框的内容，把获取文件名的代码放到了draw\_detections\_v3前面，并且在其后加上保存检测框中目标的代码：

save\_dets(im, dets, nboxes, thresh, b);

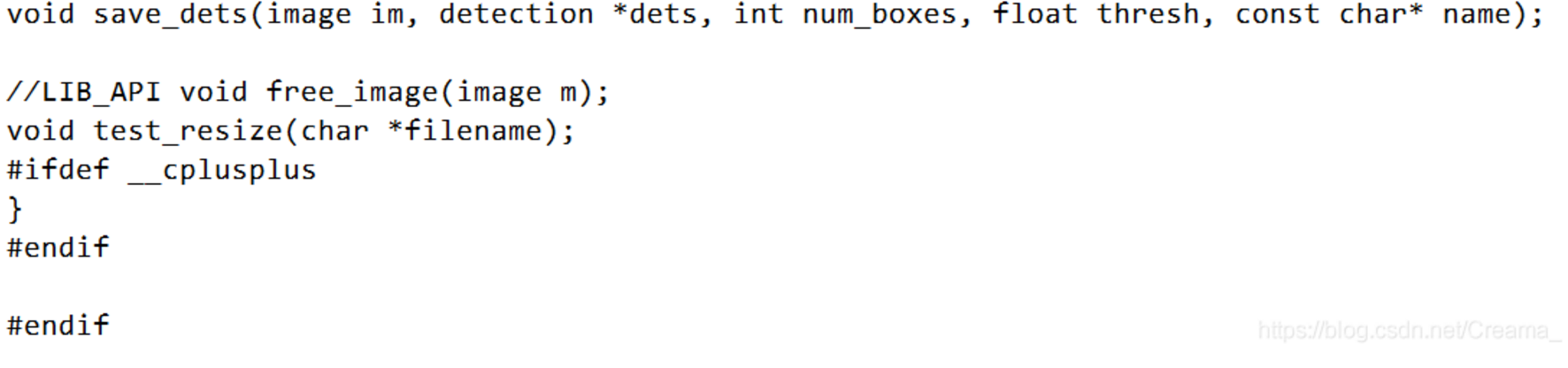
2.修改image.h和image.c

打开src/image.h

在文件末尾加上函数save\_dets的声明：

void save\_dets(image im, detection \*dets, int num\_boxes, float thresh, const char\* name);

如下图所示：



在这里插入图片描述

保存后打开src/image.c

在文件开头加入函数save\_dets的定义：

void save\_dets(image im, detection \*dets, int num\_boxes, float thresh, const char\* name){

int reg[4];

int width = im.w;

int height = im.h;

for(int i=0; i < num\_boxes; ++i){

int best\_class = -1;

float best\_class\_prob = thresh;

for (int j = 0; j < dets[i].classes; ++j){

if (dets[i].prob[j] > best\_class\_prob) {

best\_class = j;

best\_class\_prob = dets[i].prob[j];

}

}

if (best\_class >= 0){

box b = dets[i].bbox;

reg[0] = round((b.x - b.w / 2) \* width);

reg[1] = round((b.y - b.h / 2) \* height);

reg[2] = round(b.w \* width);

reg[3] = round(b.h \* height);

if (reg[0] < 0) reg[0] = 0;

if (reg[0] > width - 1) reg[0] = width - 1;

if (reg[1] < 0) reg[1] = 0;

if (reg[1] > height - 1) reg[1] = height - 1;

if (reg[2] + reg[0] > width) reg[2] = width - reg[0];

if (reg[3] + reg[1] > height) reg[3] = height - reg[1];

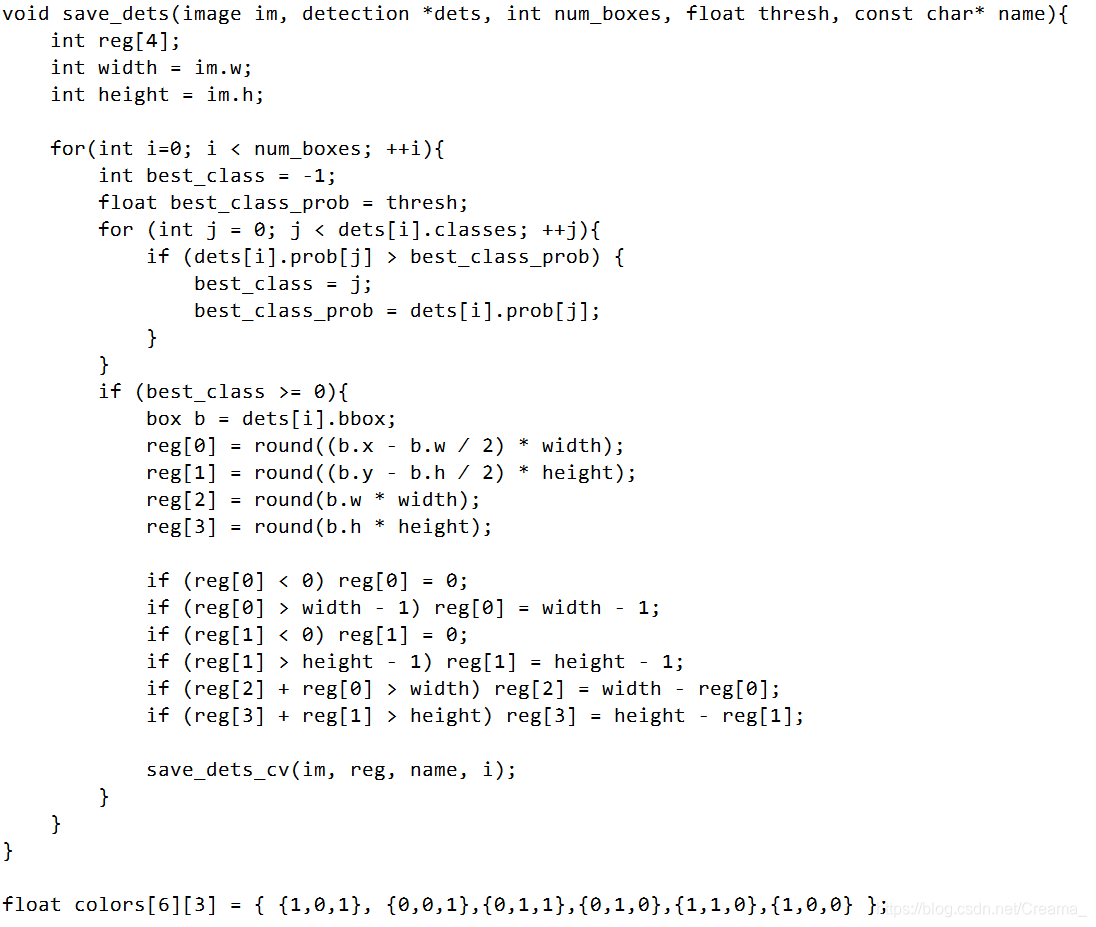
save\_dets\_cv(im, reg, name, i);

}

}

}

如下图所示：



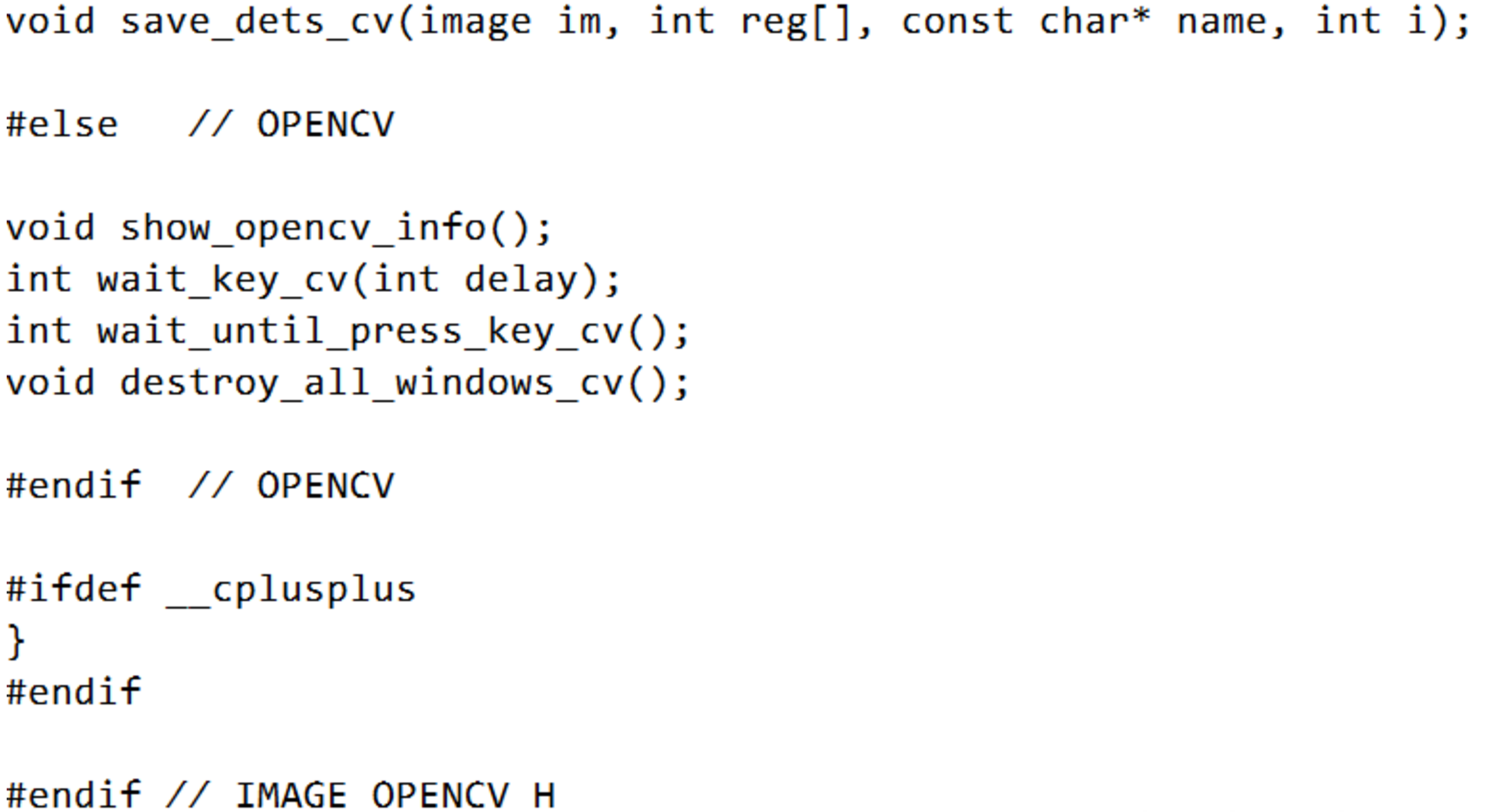
3.修改image\_opencv.h和image\_opencv.cpp

打开src/image\_opencv.h

在文件末尾加上函数save\_dets\_cv的声明：

void save\_dets\_cv(image im, int reg[], const char\* name, int i);

如下图所示：



保存后打开src/image\_opencv.cpp

在文件末尾加上函数save\_dets\_cv的定义：

void save\_dets\_cv(image im, int reg[], const char\* name, int i){

cv::Mat mat = image\_to\_mat(im);

cv::cvtColor(mat, mat, cv::COLOR\_BGR2RGB);

cv::Mat dets = mat(cv::Rect(reg[0], reg[1], reg[2], reg[3]));

//string name\_dets = name;

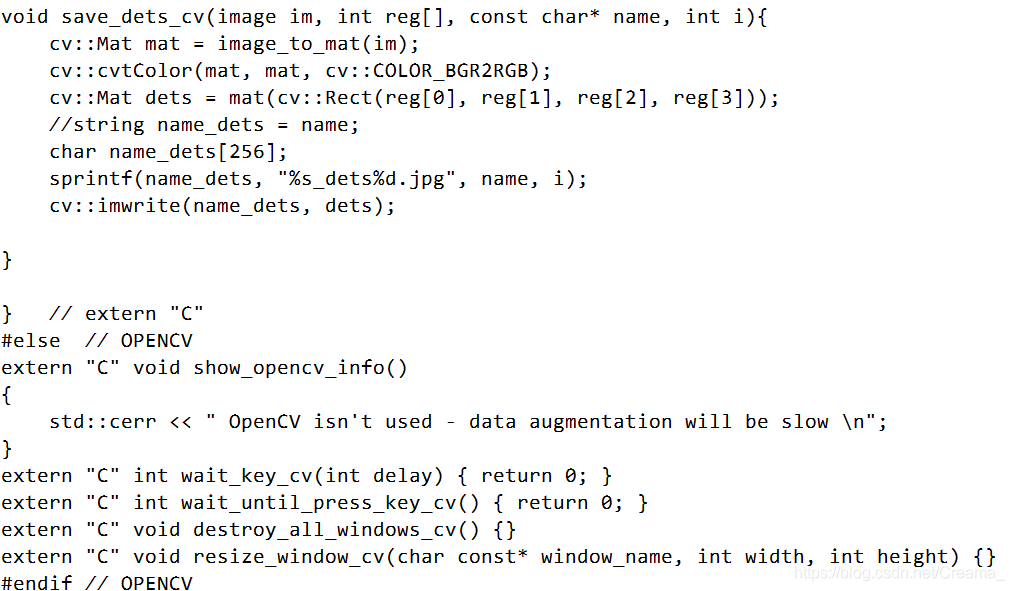
char name\_dets[256];

sprintf(name\_dets, "%s\_dets%d.jpg", name, i);

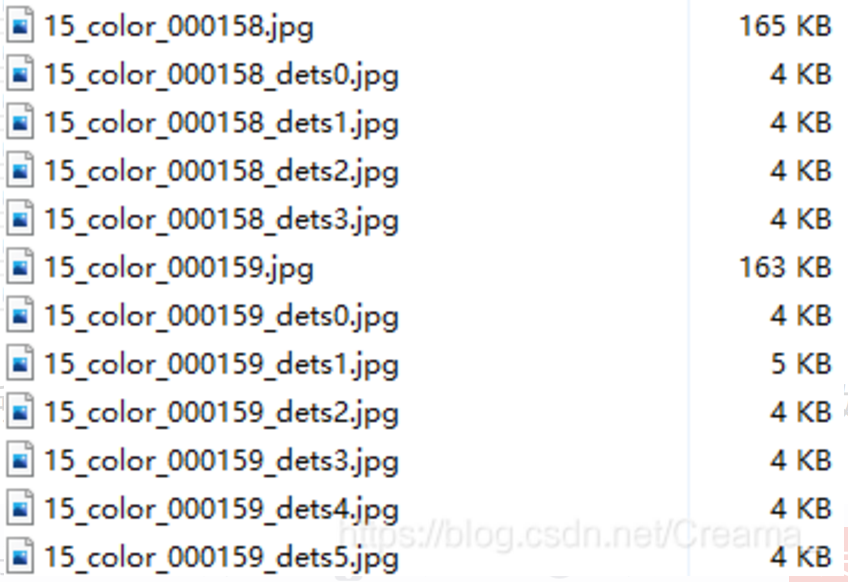
cv::imwrite(name\_dets, dets);

}

如下图所示：



到这里代码就写完了，保存所有文件，重新make。输入检测的命令行，结果会保存在output/，如下图所示，带dets的是每个检测框中的内容，如果不想保存大图可以注释掉detector.c中的save\_image。



因为C++学的不是很好，所以只能借助opencv的方法切片并保存。

# 八、显示中文标签

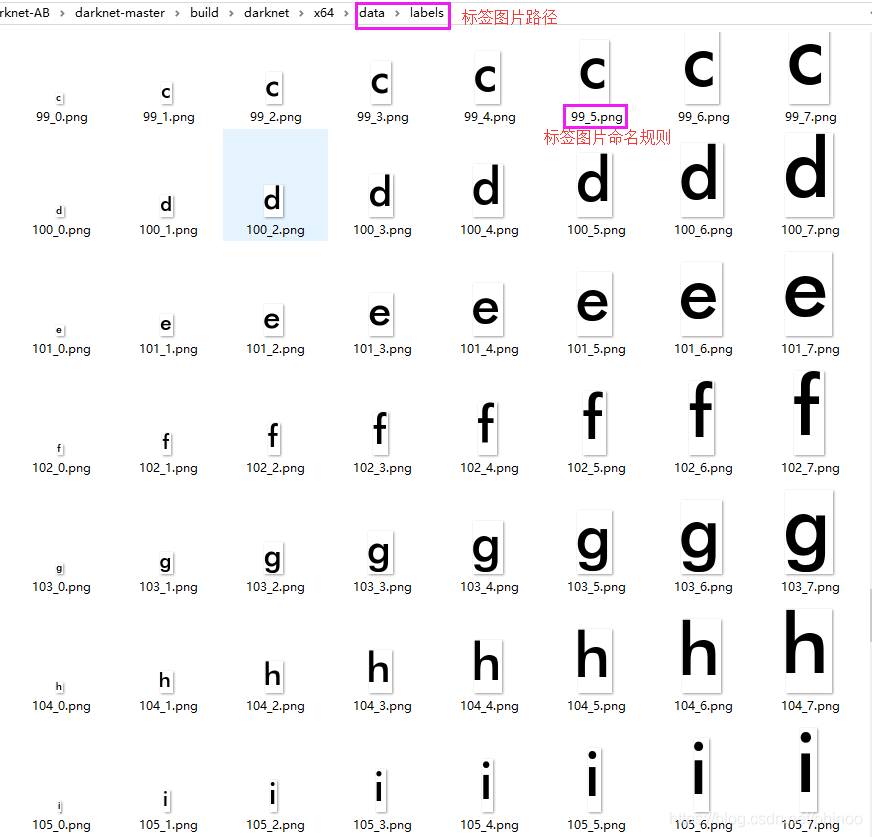
参考：https://blog.csdn.net/phinoo/article/details/89501569

第一步：生成对应的标签图片

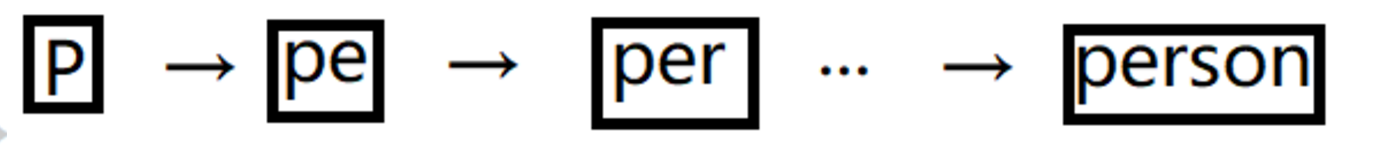
在此之前，先普及一点源码画标签的知识。假如，现在要往检测图片的某一位置绘制标签Person，则程序操作流程为：

首先，根据检测框的大小确定标签显示的大小（源码中为一组公式）

然后，根据标签中的字母拼接最终标签图，以Person为例讲解



如上图所示，源码的 darknet-master/data/labels 路径（有两个labels文件夹，Win10环境使用图中所示的labels文件夹，Ubuntu使用 darknet-master/data/labels 文件夹）下有对应大小不同的字母图片，其命名规则为：ASCII\_大小 ，其中ASCII表示字母对应的ASCII值，这样处理是为了计算方便，大小是根据检测框大小计算得到的，假定现在Person标签的大小为5号，则程序会将Person标签分解为6个字母：‘P’ ‘e’ ‘r’ ‘s’ ‘o’ ‘n’ ,首先找到‘P’字母的5号图片，然后找到‘e’字母的5号图片将其拼接在’P’字母的右边…其它字母类似，如下图所示：



最后，将生成的标签图片覆盖到检测图片的正确位置，注意是覆盖，即像素值取代?是不是很艰辛！明白了画标签的原理，要改就简单多了。现在，来生成中文标签所需的图片，这儿与原代码不同的是中文标签基于词语生成图片，如“行人”，则不拆分成“行”和“人”，主要是因为字母可以根据ASCII简便计算，但中文不行，当然，你也可以拆开来做，只是稍微复杂点，但这儿为了简单起见，就不拆开。生成标签的图片代码如下：

import os

import string

import pipes

font = 'futura-normal'

fileName=open("my.names",'r')

l=[label.lstrip().rstrip() for label in fileName.readlines()]

for i in range(8):

index=0

for word in l:

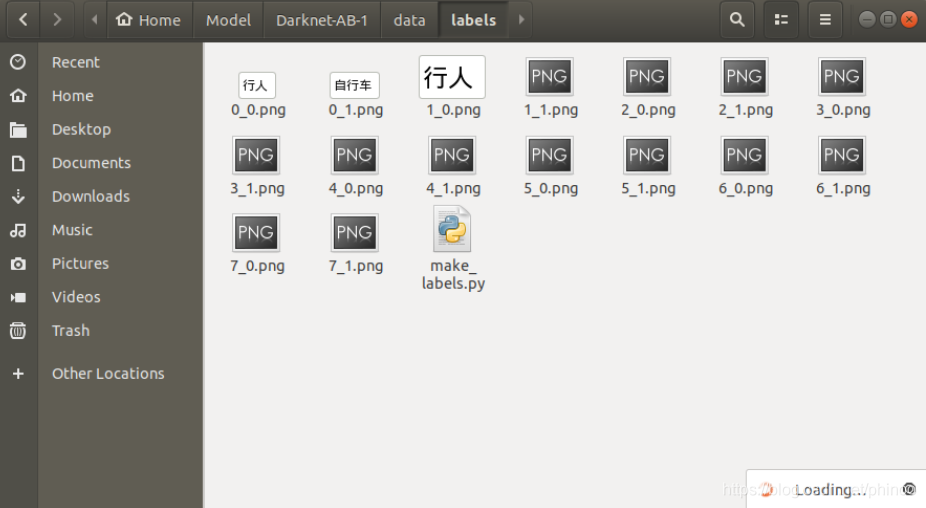
os.system(

"convert -fill black -background white -bordercolor white -border 4 -font /usr/share/fonts/truetype/droid/DroidSansFallbackFull.ttf -pointsize %d label:\"%s\" \"%d\_%s.png\"" % ((i+1)\*12, word, i,index))

index+=1

警告：本段代码只能在Ubuntu操作系统下运行，没有Ubuntu的小伙伴可以让别人替你生成好后拷贝过来。

注意：你只需要将此段代码粘贴进任何一个.py文件，放到labels文件夹下面，在该文件夹下面运行即可。此外：你需要修改代码中的“my.names”为你的标签文件，运行的结果是：在labels文件夹下面生成了一系列标签图片，如下图所示（请确认你的结果正确）：



此处：第一个数字表示大小，分为8个等级，0-7表示，第二个数字表示标签编号，和\*\*.names文件对应。

第二步：修改部分代码

Step1 修改 src/image.c 中的部分代码

查找 get\_label\_v3函数的声明处，在其下面添加函数 get\_label\_v3\_my ，采用添加不采用修改的方式主要是避免破坏源码，其它地方的修改类似，函数代码为：

image get\_label\_v3\_my(image \*\*characters, char \*labelindex, int size)

{

size = size / 10;

if (size > 7) size = 7;

image label = make\_empty\_image(0, 0, 0);

int class,i=0,nlabels=1;

int len=strlen(labelindex);

for(i=0;i<len;i++)

{

if(labelindex[i]==',') ++nlabels;

}

for(i=0;i<nlabels;i++){

class=atoi(labelindex);

image l = characters[size][class];

image n = tile\_images(label, l, -size - 1 + (size + 1) / 2);

free\_image(label);

label = n;

labelindex=strchr(labelindex,',')+1;

}

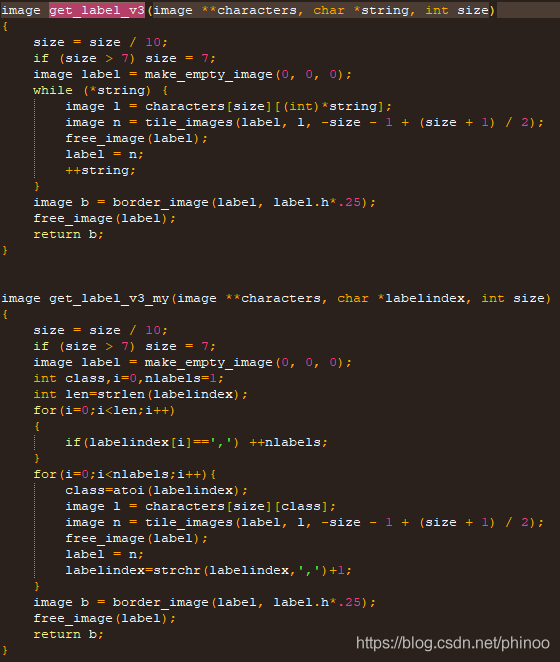
image b = border\_image(label, label.h\*.25);

free\_image(label);

return b;

}

添加代码过后，你的文件应该如下图所示，其中具体意义这里不进行讲解，感兴趣的小伙伴自行研究和修改。



step2 修改对get\_label\_v3的调用为get\_label\_v3\_my

找到 image.c 中的 draw\_detections\_v3函数。然后找到 if ( alphabet ) { … } 代码块，将其改为：

if (alphabet) {

char labelindex[100]= { 0 };

char class[10]={0};

sprintf(class,"%d",selected\_detections[i].best\_class);

strcat(labelindex, class);

int j;

for (j = 0; j < classes; ++j) {

if (selected\_detections[i].det.prob[j] > thresh && j != selected\_detections[i].best\_class) {

strcat(labelindex,", ");

sprintf(class,"%d",selected\_detections[i].best\_class);

strcat(labelindex, class);

}

}

image label = get\_label\_v3\_my(alphabet, labelindex, (im.h\*.03));

draw\_label(im, top + width, left, label, rgb);

free\_image(label);

}

当然，这儿你也可以新建函数，然后改函数调用，从而避免破坏源码。此处修改是为了调用上面定义的函数。

step3 在 image.c文件的 load\_alphabet 函数下面添加如下函数：

image \*\*load\_labels(int classes)

{

int i, j;

const int nsize = 8;

image\*\* alphabets = (image\*\*)calloc(nsize, sizeof(image\*));

for(j = 0; j < nsize; ++j){

alphabets[j] = (image\*)calloc(classes, sizeof(image));

for(i = 0; i < classes; ++i){

char buff[256];

sprintf(buff, "data/labels/%d\_%d.png", j, i);

alphabets[j][i] = load\_image\_color(buff, 0, 0);

}

}

return alphabets;

}

本函数的作用为将前面我们生成的标签图片加载进内存，保存在二维结构体数组中

step4 在src/image.h 的 image \*\*load\_alphabet();行下面添加下面行：

image \*\*load\_labels(int);

此行代码的作用为声明方法。

stpe5 修改 src/detector.c 中的函数调用，使之调用刚才添加的 load\_labels

找到 test\_detector 函数的定义，将 image \*\*alphabet =load\_alphabet()； 修改为：

image \*\*alphabet = load\_labels(names\_size);

第三步：重新编译，直接回到根目录make即可

到此，教程完毕，其它诸如调节字体大小和框粗细的问题都可以在image.c函数中进行修改。本人对Darknet框架作了精细的研究，并进行了大量的注释。 <https://github.com/Eric3911/yolov3_darknet>