使用YOLO

使用环境：Centos7.3 ，CUDA 9.0，

GPU：NVIDIA GM107GL[Quadro k620] （专业绘图）

一 安装

1. 安装opencv（下载源码包安装）

我安装的版本是2.4.10，安装教程去官网查找，需要安装依赖。

网址：<https://opencv.org/>

步骤

a）安装依赖

--->  添加源

sudo yum -y install epel-release

--->gcc,g++等这些基本的工具,一般都有，没有的话再安装

sudo yum -y install gcc gcc-c++ cmake python-devel numpy

--->然后是opencv的依赖项

sudo yum -y install gtk2-devel gimp-devel gimp-devel-tools gimp-help-browser zlib-devel libtiff-devel libjpeg-devellibpng-devel libavc1394-devel libraw1394-devel libdc1394-devel jasper-devel jasper-utils swig python libtool nasm libv4l-devel gstreamer-plugins-base-devel

b）编译opencv

按照官网步骤来

C）参考网址

<http://blog.csdn.net/kakitgogogo/article/details/52490010>

<https://www.cnblogs.com/hxbbing/p/4894813.html>

d）总结

注意: Anconda 自带是python-opencv，不属于opencv源码库， 加载其路径是不可以的，必须要编译opencv源码，生成库文 件和添加环境变量。

网址：<http://www.cnblogs.com/wyuzl/p/7889007.html>

安装的时候需要一大堆依赖（宁可信其有），且官网ubuntu的依赖和centos的依赖有所区别，要注意。

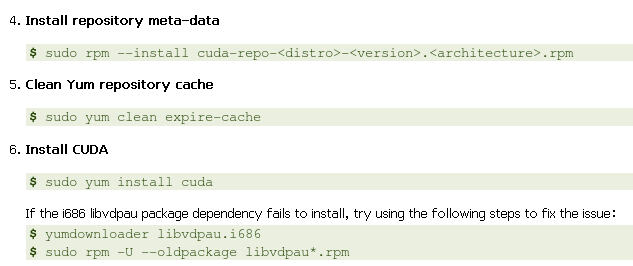
1. 安装CUDA（同上）

我安装的是cuda9.0，从网址下载好工具包后，直接用官网的命令安装。

下载网址：<https://developer.nvidia.com/cuda-zone>

安装网址：

<https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html#package-manager-installation>



添加环境变量：在～/ 目录下的.bashrc文件中添加

export CUDA\_HOME=/usr/local/cuda-9.0

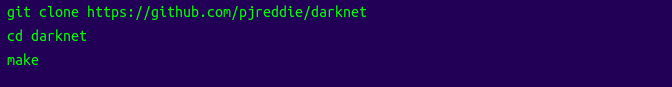
export LD\_LIBRARY\_PATH=/usr/local/cuda-9.0/lib64:$LD\_LIBRARY\_PATH

export PATH=/usr/local/cuda-9.0/bin:$PATH

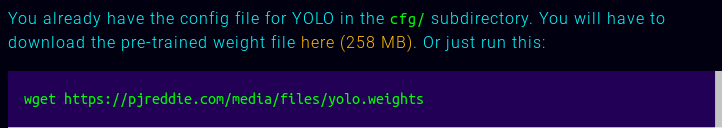
（3）安装Yolo

按照官网教程：<https://pjreddie.com/darknet/yolo/>

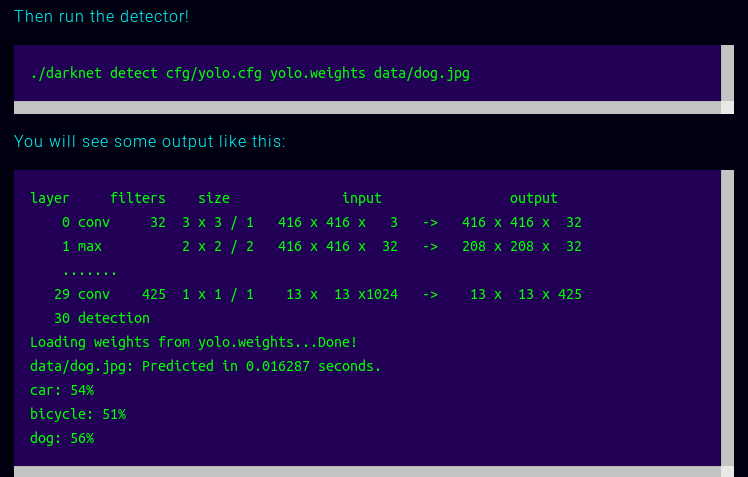
First：安装源码包：



Second：下载训练好的权重文件



Three：识别



Q：-->make找不到opencv路径的问题

解决网址：<http://www.cnblogs.com/wyuzl/p/7889007.html>

一般不会出现这个原因，因为我的电脑卸载过自带的opencv，然后有自己编译过另一个版本的opencv，所以才会有这个错误。

1. 安装CUDNN（一个深度学习算法加速器）

我目前没有安装，看后续实测效果再决定。

二 测试问题：

(1)Q: --->一直没有在pre-pic 中画出框框。

答：我猜是因为我没有训练神经网络，因为刚接触，而且官网没有给出训练数据的步骤。我猜的是不对的。

(2)Q: --->这里，我们所有的准备工作就完成了，接下来就可以开始训练了，这里有两种情况：

 A.使用yolo官网提供的已经训练好的模型，初始化，进行训练：

      ./darknet partial cfg/darknet19\_448.cfg darknet19\_448.weights darknet19\_448.conv.23 23

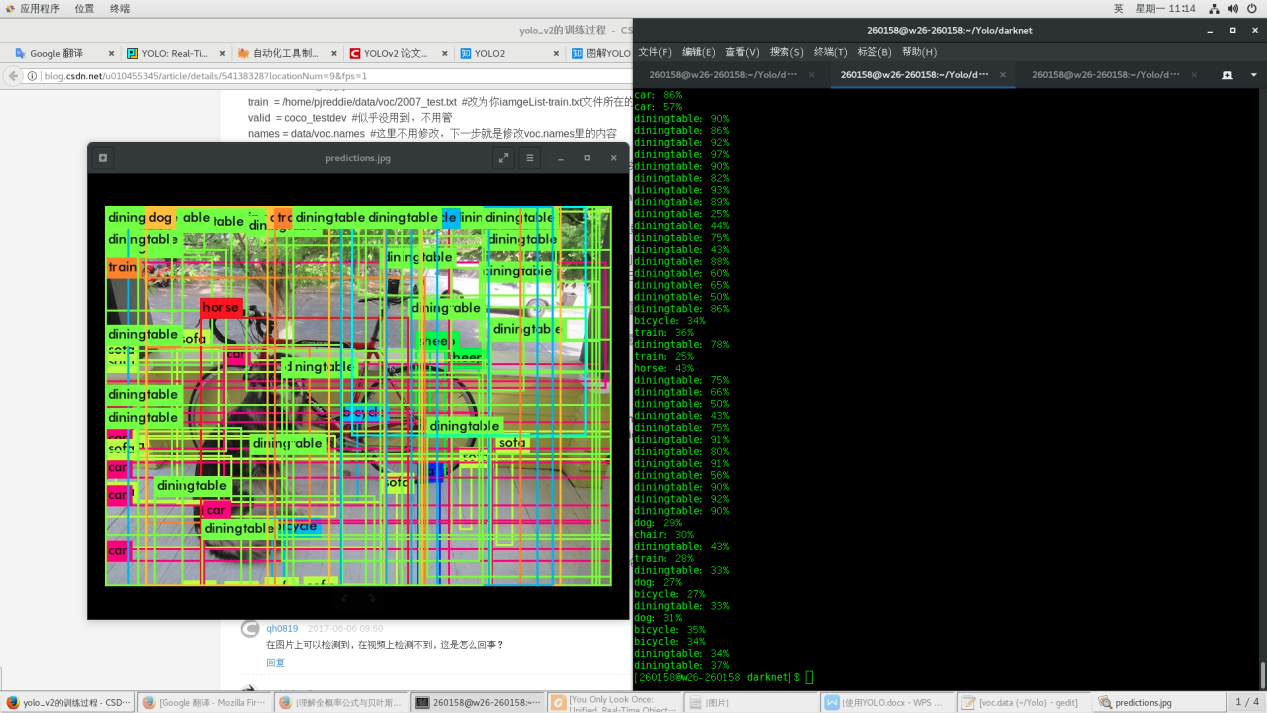
     ./darknet detector train cfg/voc.data cfg/yolo-voc.cfg darknet19\_448.conv.23

   B.用自己的数据集直接进行训练：

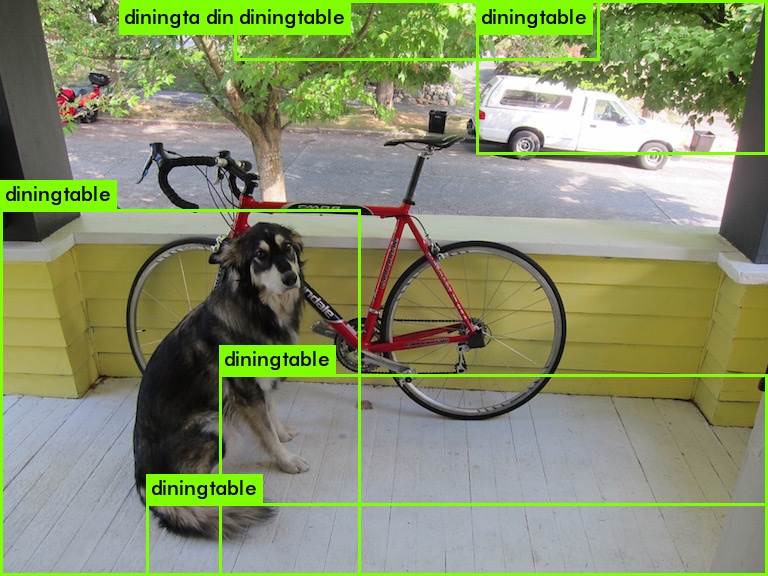
./darknet detector train cfg/voc.data cfg/yolo-voc.cfg

(3)Q: --->执行完上面这两步后，会出现标注，但是满屏幕都是标注，出现这个原因是阈值过小，提高就会OK，但是标注的内容是不对的。

如下图：

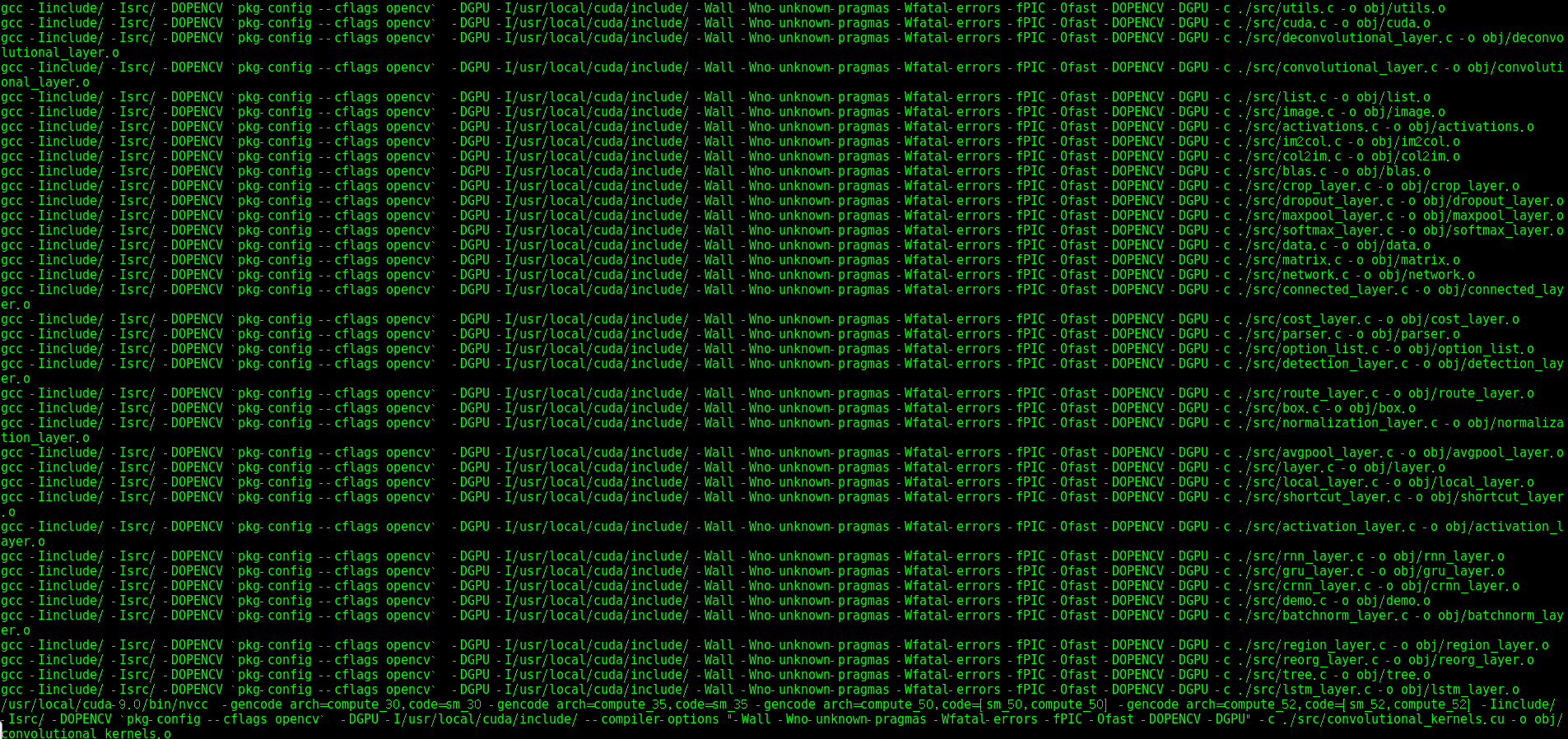


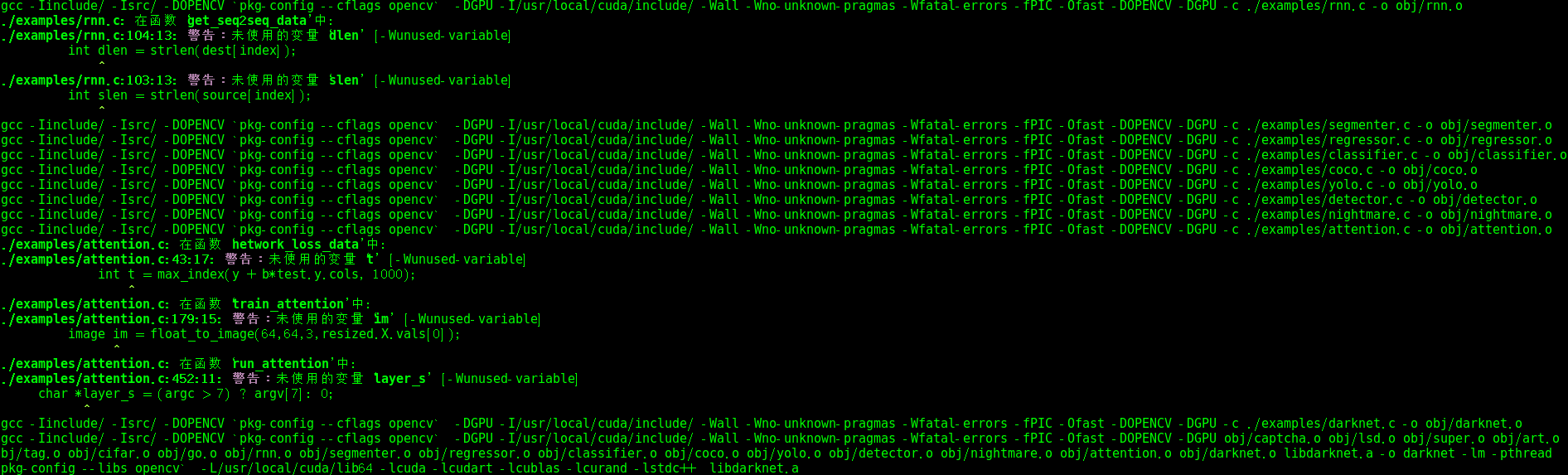
改变阈值后如下图：



(4)Finally：最终解决的方法（在成功的时候有砸电脑的冲动）

我重新安装了一遍darknet,make的时候出现以下信息：

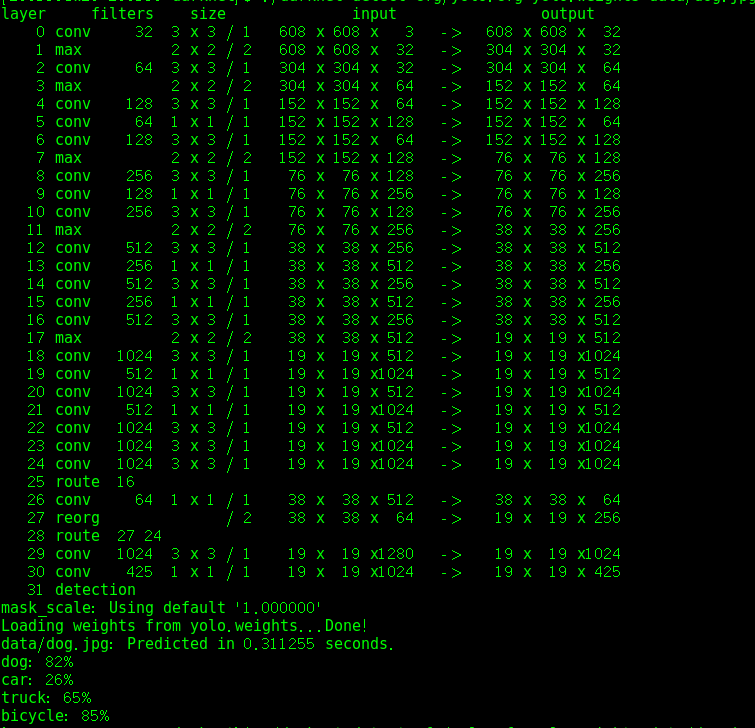


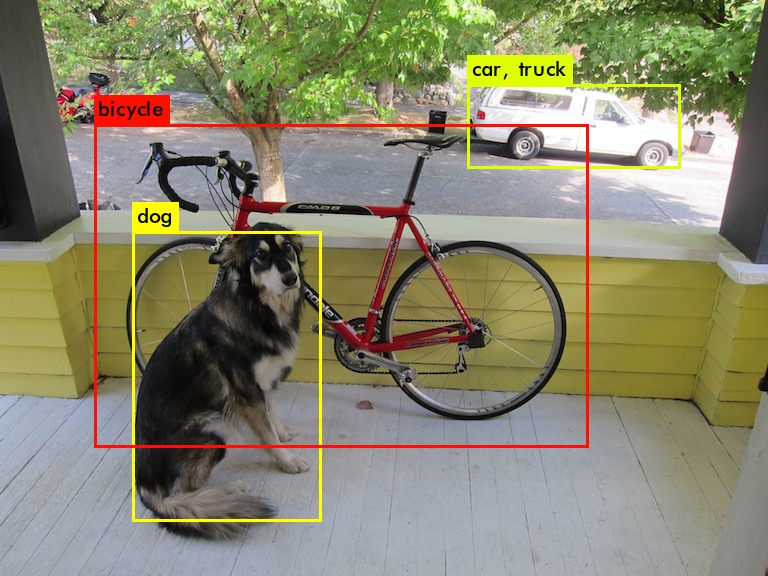


应该是安装成功了。

然后直接执行：  
./darknet detect cfg/yolo.cfg yolo.weights data/dog.jpg

就可以成功标注，如下图：

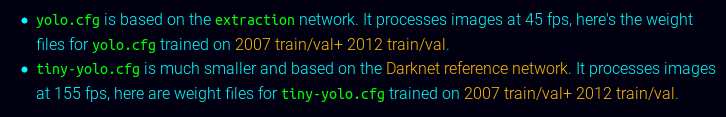




像 voc.data,voc.names,还有训练数据集，测试数据集之类的文件全都没有改。

.weights是已经根据训练集 voc2007 or voc2012 训练好的。

.cfg文件对应的如下图：

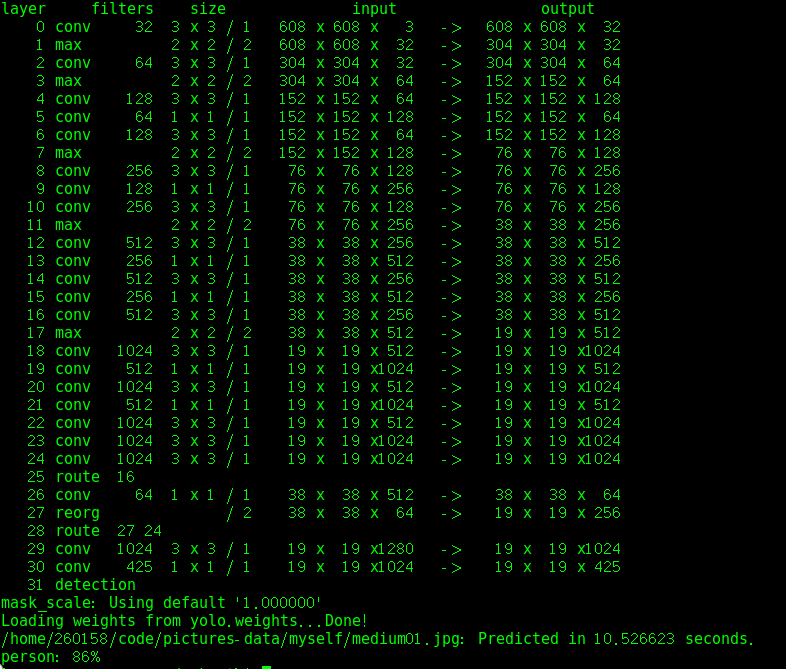


（5）比较CPU 和 GPU 版本的差别

在重新编译了没有GPU版本的darknet之后，对同一张图片进行测试，结果如下：

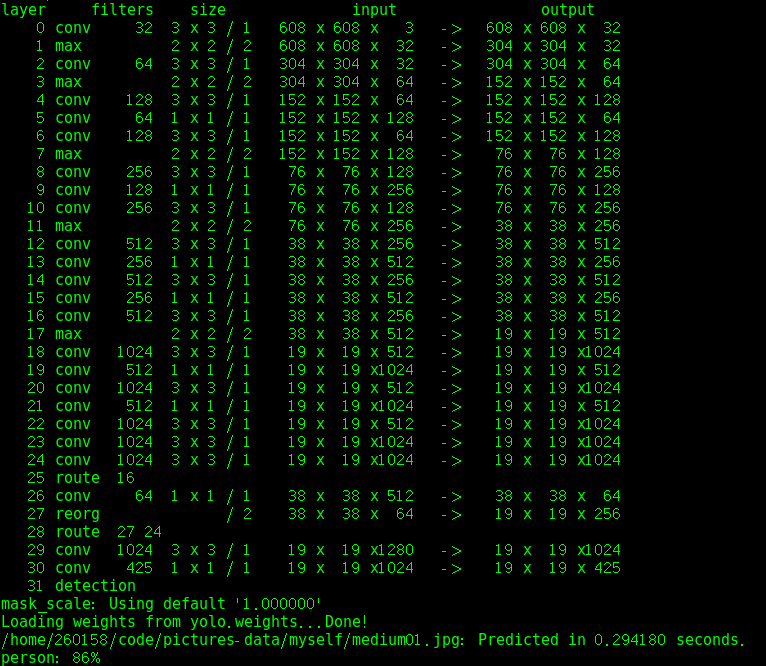


--->CPU版本：



耗时 10.526s

--->GPU版本：

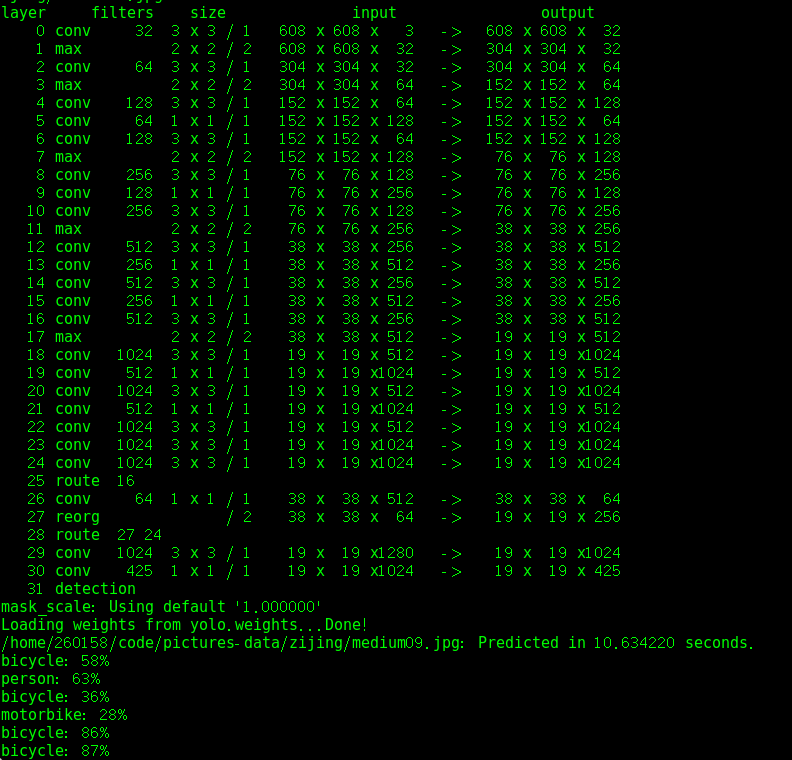


耗时：0.2941s

再来测试一张图片：

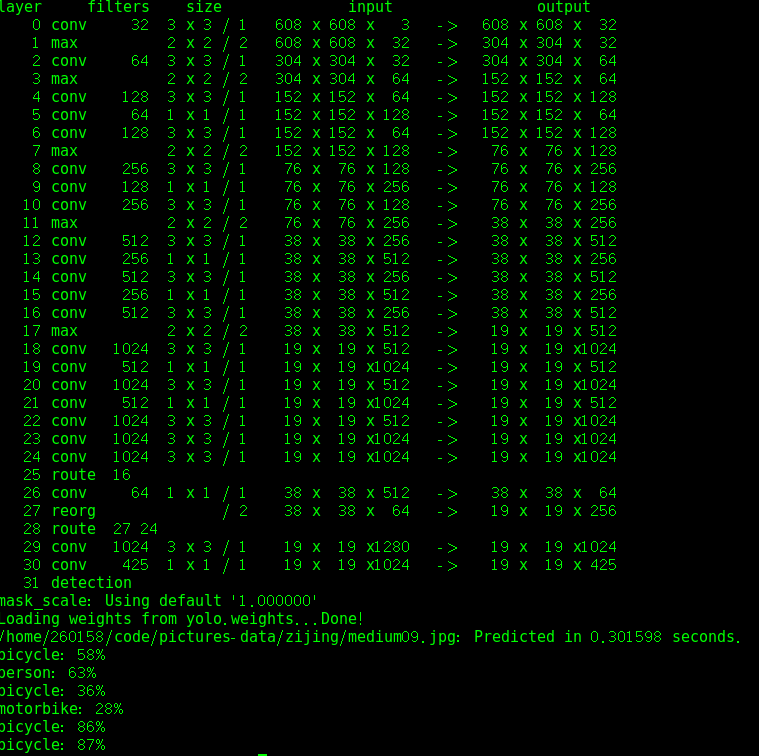


--->CPU版本：



耗时 10.634s

--->GPU版本：



耗时：0.301s

没有对比就没有伤害，不过这也差太多了。。。。。。突然想到会不会加了CUDNN也会差很多？？？？

三 训练自己的数据集

Darknet是一个用C和CUDA编写的开源神经网络框架。它安装速度快，易于安装，并支持CPU和GPU计算。Yolo是其框架的具体实现方法。Yolo是物体检测的算法。

Caffe，tensorflow等框架也可以用Yolo的方法来实现。

tensorflow版本源码

<https://zhuanlan.zhihu.com/p/25053311>

caffe版本源码

<http://blog.csdn.net/u014380165/article/details/72553074>

1、yolo 和tiny-yolo的关系

tiny-yolo是更快的模型，但是准确度比较低。

2、yolo-v2只识别person 这一类

我修改的是home/260158/darknet 这个目录下的内容

## ****--->修改源代码****

**（1）修改cfg/voc.data**

**classess=20    改成 classes = 1**

**（2）修改data/voc.names**

**只留下person这一类**

**（3）修改examples/detector.c**

void run\_detector(int argc, char \*\*argv)//该函数中，倒数第三行.line=5422

int classes = option\_find\_int(options, "classes", 1);//最后一个参数修改成1

void test\_detector(char \*datacfg, char \*cfgfile, char \*weightfile, char \*filename, float thresh)函数中5

draw\_detections(im, l.w\*l.h\*l.n, thresh, boxes, probs, names, alphabet, 1); //最后一个参数修改成1。line=481

## ****--->重新编译和测试****

**（1）重新编译**

**make clean**

**make**

**（2）测试**

**./darknet detect cfg/yolo.cfg yolo.weights /home/260158/pictures-data/mark-pic/video1/images/491.jpg**

**上面为我的图片路径**



**--->YOLO选择出只有某一类的检测框并将框保存在txt中**

<http://blog.csdn.net/zhangxinyu11021130/article/details/76473193?locationNum=10&fps=1>

3、测试识别视频的效果

（1）./darknet detector demo cfg/yolo.cfg yolo.weights

这条命令是打开摄像头进行实时检测

（2）./darknet detector demo cfg/yolo.cfg yolo.weights <video file>

这条命令是检测视频，视频要放在darknet的根目录下面。

我这里是测试视频文件。

4、准备自己的voc型数据集

（1）将图片的名称命名为“000001.jpg”的格式,并将名称保存到train.txt文 件夹下

程序在/code 目录下的rename.py

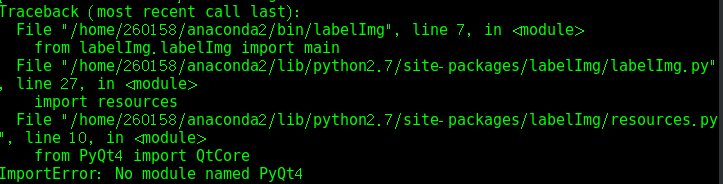
（2）安装labelmg

--->pip install labelImg

安装的是1.5.2，运行时提示没有PyQt4，下一步安装

--->sudo yum install PyQt4

安装成功后，还是有提示错误



--->conda install pyqt=4

运行成功，Conda是一个包管理的工具

--->labelImg

成功打开界面

（3）打标数据

使用方法:

--->修改默认的XML文件保存位置，使用快捷键“Ctrl+R”，改为自定义位置，这里的路径一定不能包含中文，否则无法保存。

--->源码文件夹中使用notepad++打开data/predefined\_classes.txt，修改默认类别，比如改成person、car、motorcycle三个类别。

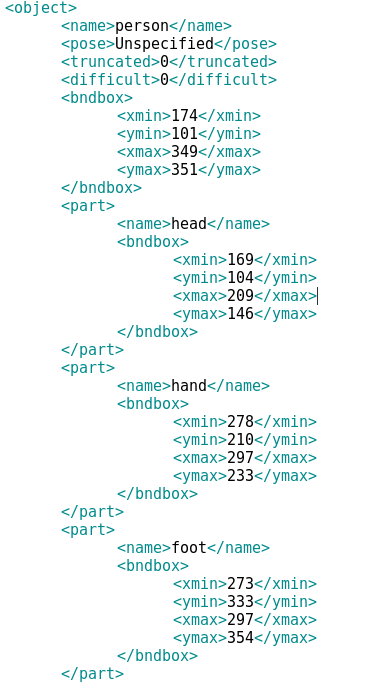
---> “Open Dir”打开图片文件夹，选择第一张图片开始进行标注，使用“Create RectBox”或者“Ctrl+N”开始画框，单击结束画框，再双击选择类别。完成一张图片后点击“Save”保存，此时XML文件已经保存到本地了。点击“Next Image”转到下一张图片。

--->标注过程中可随时返回进行修改，后保存的文件会覆盖之前的。

--->完成标注后打开XML文件，发现确实和PASCAL VOC所用格式一样。

成功完成打标。

Q: 比如标注人的类别，这个类别里有head，hand，foot等部分，下载的数据的xml中会有分开打标，但labelImg里没有这个功能。如下图：



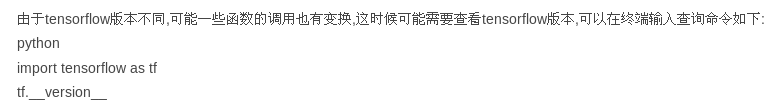
5、测试yolo-tensorflow版本

（1）安装tensorflow

因为我之前已经安装过（用pip命令可以直接安装），或者自己上网去找教程，笔较简单。

（2）查看tensorflow版本

我的是1.3.0版本的



2017-11-29 19-01-28 的屏幕截图

（3）测试

6、测试yolo-caffe版本

（1）安装caffe

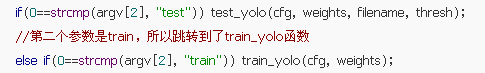
四YOLO原理理解

1、源码结构

（1）首先从main()函数开始看，该函数在darknet.c，在/darknet/examples/文件夹下。

该函数主要是命令行第一个输入参数的匹配，如果匹配到则直接跳到该函数。相当于算法框架的入口。

（2）在/darknet/examples/下的yolo.c文件，run\_yolo()函数

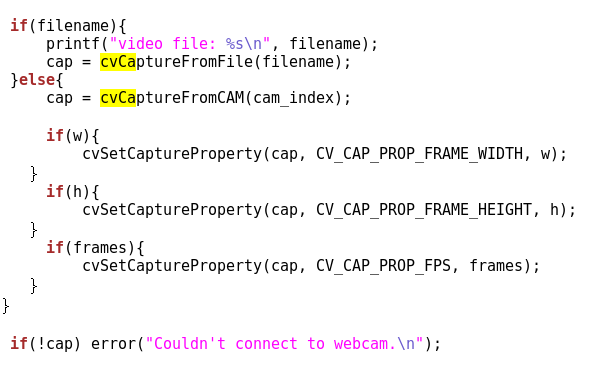


该文件主要是使用darknet 实现YOLO论文的思想。

我们现在是要训练自己的数据集，所以第二个参数是train.

（3）在/darknet/src/下的demo.c文件,demo()函数

主要实现yolo处理视频文件

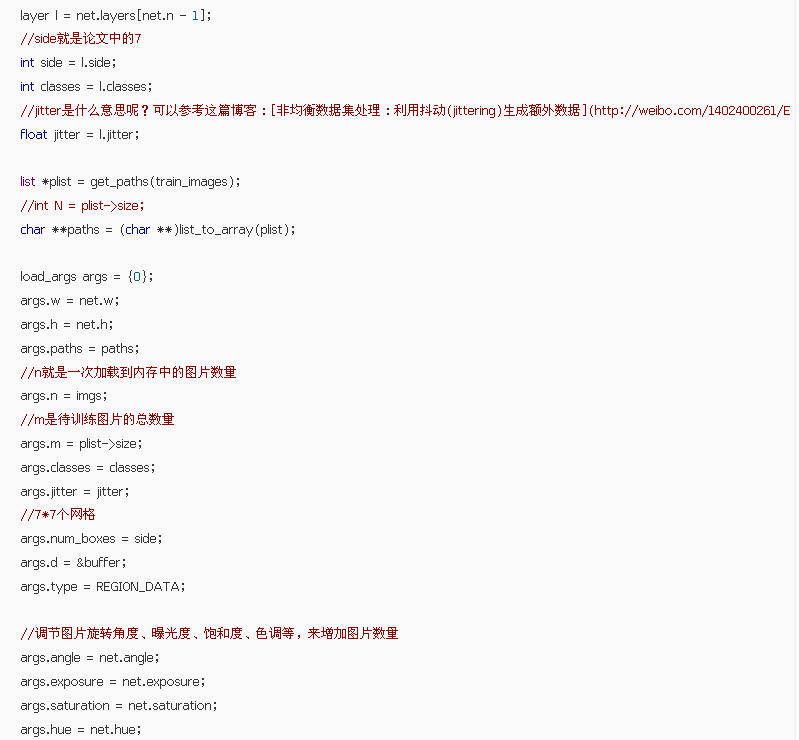


（4）在/darknet/examples/下的yolo.c文件，train\_yolo()函数



第三个参数是cfg/yolo.cfg。

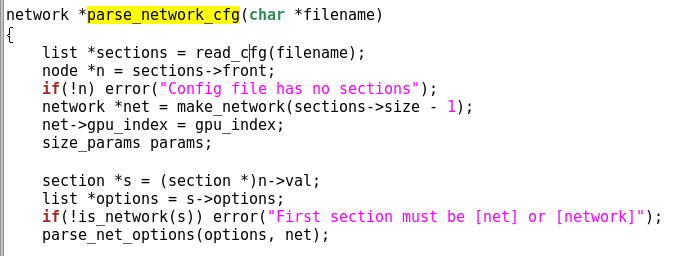
主要是加载神经网络架构和预训练参数，该函数的俩个参数指针指向这俩个文件。其余部分是对图像的一些预处理过程。



了解到这里就大致可以训练自己的数据了，下面是框架的大概实现过程。

（4）在/darknet/src/下的parser.c文件，网络参数解析函数

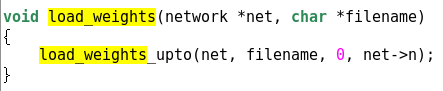
parse\_network\_cfg()



read\_cfg这个函数将所有的参数读到一个图中，这里的图可以理解为参数网，然后根据一定的规则去读这个参数网里的参数。即给参数定了一个读取的规则，这样就不会导致数据混乱。

（5）在/darknet/src/下的parser.c文件,加载权重函数

load\_weights()

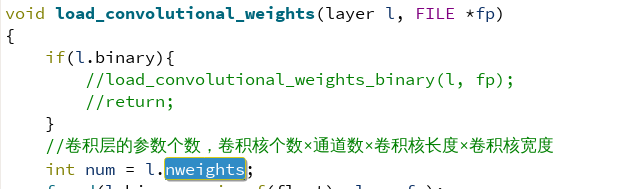


大概是这样，具体的函数没有去分析。

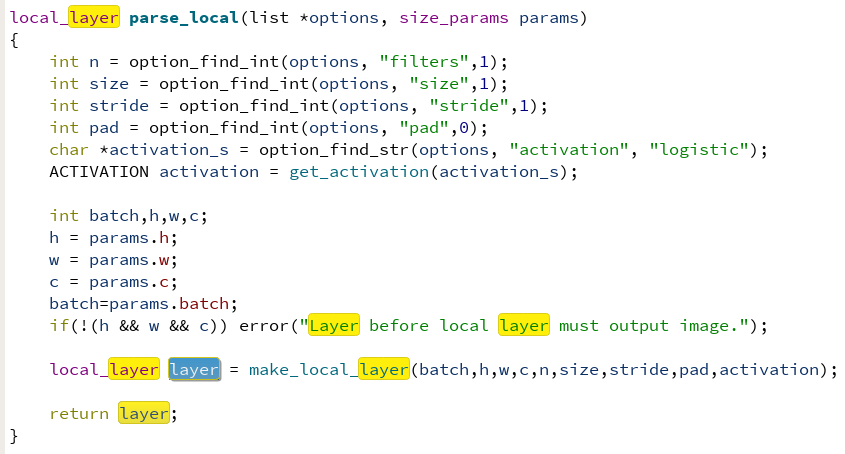
可见主要实现过程是调用load\_weights\_upto()函数



下面我们看卷积层的权重加载load\_convolutional\_weights()



下面看看如何得到结构体layer



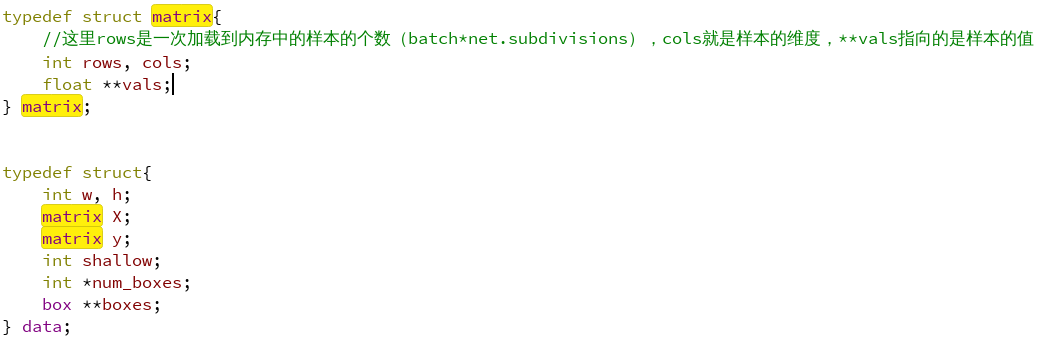
可见，这个函数返回一个结构，其包含batch等元素。

（6）在/darknet/src/下的network.c文件中，训练神经网络函数

train\_network(network \*net, data d)

注意：yolov2的大部分结构体定义都在 /include/目录下的darknet.h中。

先看下data和matrix结构体，我猜是打标好的数据结构

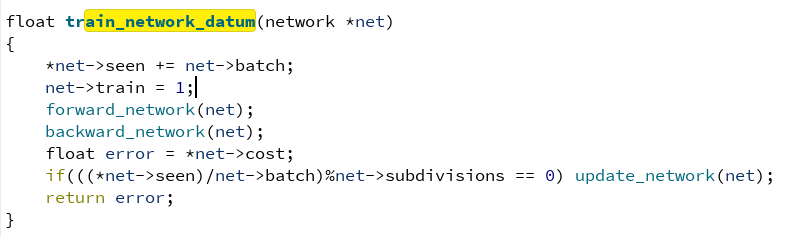


下面看train\_network()函数内容。



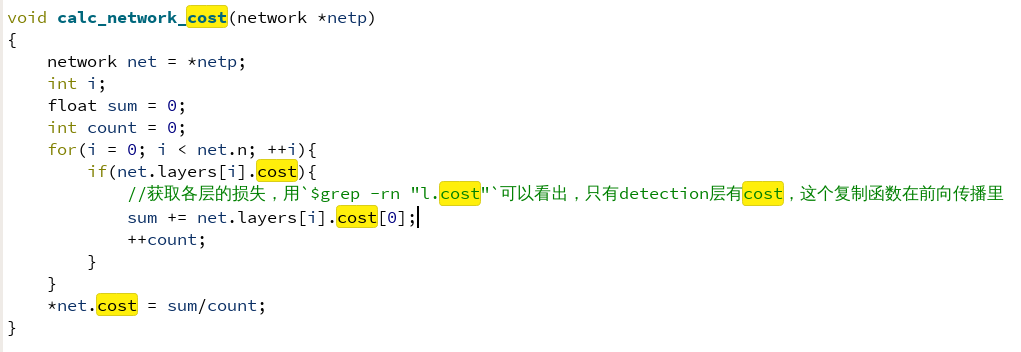
然后分析以上函数调用的其它函数，下面是data.c内容





训练主要是forward\_network和backward\_network,这俩个函数。

下面是计算代价函数



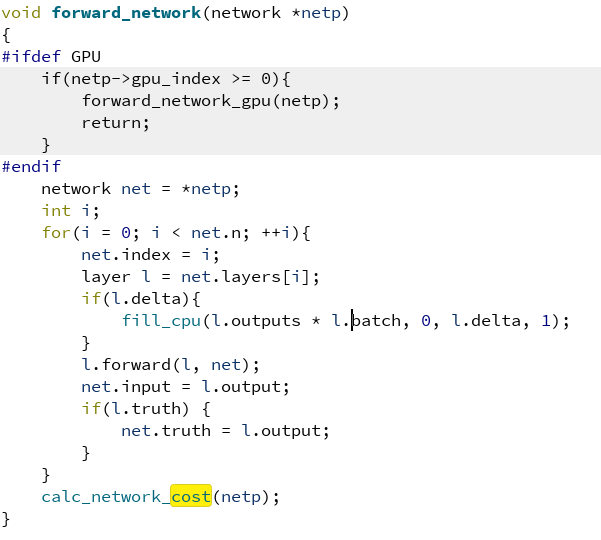
该函数最后返回平均损失。

到这里我们只剩下最重要的两个函数：

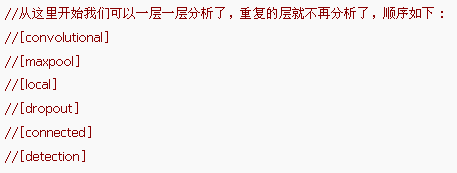
forward\_network(network \*netp); //前向传播

backward\_network(network \*netp); //反向传播

（7）在src/network.c文件，前向传播函数forward\_network(network \*netp)



分为GPU和CPU俩种情况，主要是forward\_network\_gpu()和forward()函数，下面分析每一层



由于还没看懂，所以先附上网络的解释。

<http://blog.csdn.net/u014540717/article/details/53232426>

（8）在src/network.c文件，反向传播函数backward\_network(network \*netp)

同样附上网址：

<http://blog.csdn.net/u014540717/article/details/53485680>