# 车辆定位（Windows Build）

[主页](http://lxiongh.com/)[分类](http://lxiongh.com/categories/)[标签](http://lxiongh.com/tags/)[链接](http://lxiongh.com/links/)[关于](http://lxiongh.com/about/)

【原创声明，转载请注明出处】<http://lxiongh.com/2015/04/24/Caffe_Windows_Detection_Build/>

本人修改的 caffe detection (windows)（[gitcafe 链接](https://gitcafe.com/lxiongh/Caffe_Windows_Detection)，[github 链接](https://github.com/lxiongh/Caffe_Windows_Detection)），针对车辆检测编译后的结果，目录中的主要文件如下所示。（注：本机为Window 7 x64 位系统，GTX 650 GPU 显卡。在无显卡机上请使用CPU模式运行）

images/ -> 存储了一些测试用图片

base.prototxt -> 网络结构定义，匆删除匆修改

car\_mean.binaryproto -> 车辆均值文件，匆删除匆修改

model.caffemodel -> 训练好的模型参数

det\_net.exe -> 车辆检测主程序

easy\_go.py -> python 粘接检测过程，使检测更友好，建议使用

\*.dll -> 一些必要的动态运行库

编译后的版本与模型文件过大，压缩后的大小为 374.2M，所以上传至了百度网盘<http://pan.baidu.com/s/1kT1OQTT>，可自行前往下载。相应代码修改可参见 [博文: Caffe Window 版本](http://lxiongh.com/2015/04/23/Caffe_Windows_Detection/)

# det\_net.exe

由 [Caffe\_Windows\_Detection](https://github.com/lxiongh/Caffe_Windows_Detection) （[gitcafe 链接](https://gitcafe.com/lxiongh/Caffe_Windows_Detection)，[github 链接](https://github.com/lxiongh/Caffe_Windows_Detection)）源码中的 ./tools/test\_net.cpp 编译得到。用法为：

det\_net.exe net\_model pretrained\_net\_proto iterations output\_dir

[DRAW/LABEL/BOTH] [GPU/CPU] [Device ID]

其中 net\_model 参数为训练好后的模型，即此处为 model.caffemodel，pretrained\_net\_proto参数为模型的结构定义文件，此处为 base.prototxt。另外可选参数 [DRAW/LABEL/BOTH] 不同设置分别对应『图像上画检测框并输入』、『仅输出检测的上下角点值（x1, y1, x2, y2）』、『同时输出前两者信息』

注意：如果用 det\_net.exe 作检测，文件 base.prototxt 里的 source 及 meanfile 需要做相应的修改，指定到其绝对路径。但不建议这么做，因为提供了 easy\_go.py 这个更加简便的方法调用 det\_net.exe 作车辆检测，其隐藏了一些调用细节。

# easy\_go.py

主要完成需要测试的目录图片文件的检索，临时生成 imlist.txt文件。以 base.prototxt 为原本，自动修改 source 与meanfile 对应的值，临时生成相对应的模型结构文件 exp.prototxt，供 det\_net.exe 调用。easy\_go.py 用法如下（在easy\_go.py 目录下进行）：

usage: easy\_go.py [-h] im\_dir out\_dir [{DRAW,LABEL,BOTH}] [{GPU,CPU}] [num]

positional arguments:

im\_dir the directory of input image

out\_dir the output directory

{DRAW,LABEL,BOTH} type of output {BOTH, DRAW, LABEL}

{GPU,CPU} choose GPU or CPU

num the num of image to be test

optional arguments:

h, help show this help message and exit

Enjoy it!

调用代码：

python easy\_go.py images result

**Caffe Window 代码修改（适用于车辆定位）**

[主页](http://lxiongh.com/)[分类](http://lxiongh.com/categories/)[标签](http://lxiongh.com/tags/)[链接](http://lxiongh.com/links/)[关于](http://lxiongh.com/about/)

【原创声明，转载请注明出处】<http://lxiongh.com/2015/04/23/Caffe_Windows_Detection/>

修改后适应于车辆检测的代码地址为：[https://github.com/lxiongh/Caffe\_Windows\_Detection](http://github.com/lxiongh/Caffe_Windows_Detection) 或者 gitcafe 地址<https://gitcafe.com/lxiongh/Caffe_Windows_Detection>

## Caffe Window 编译

[caffe](http://caffe.berkeleyvision.org/) 原生代码只支持Linux，并没有 window 版本下的相关实现。 由于一个项目中需要用到车辆的检测与定位（已在 linux 下利用 [修改的 caffe](http://lxiongh.com/2015/02/04/car_localization/) 代码实现）， 而项目本身的代码至今还没有移植到 linux 下（*无力吐槽师弟这效率*）。 自己动手，丰衣足食。 Google 了下，发现在 caffe 官网下其实给出了第三方的 [window caffe](https://github.com/niuzhiheng/caffe) 版本，由 [niuzhiheng](https://github.com/niuzhiheng/) 这位牛人修改得到。真是天无绝人之路，得来全不费功夫（*其实也费了些功夫，主要在修改 niuzhiheng 的代码适用于自己的车辆定位问题*）。

## [caffe window](https://github.com/niuzhiheng/caffe) 的环境及其依赖

1. windows-64 bit
2. MS Visual Studio 2012
3. CUDA Tooltik 6.5
4. 其他第三方库（如 opencv, boost, etc）

niuzhiheng 给出的第三方库[链接](http://dl.dropboxusercontent.com/u/3466743/caffe-vs2012/dependency-20140804.7z)已经无法打开，可能是 GFW 在作恶。本人在百度网盘里上传了一个[第三方库](http://pan.baidu.com/s/1jG5o7Am)备份（*小插曲：从百度网盘中搜索关键字 dependency-20140804.7z，结果发现是杨师兄上传的，缘分啊~~~*）。

**编译步骤**

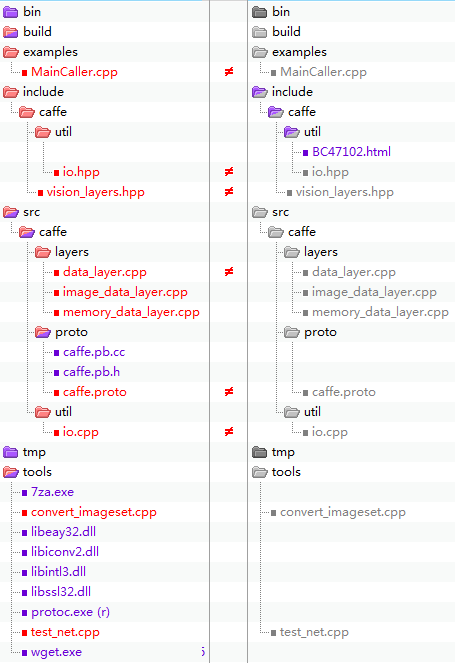
为了判断环境及依赖是否正确，利用 niuzhiheng 没有修改过的源码进行编译，先排除由于环境或者依赖不正确带来的编译问题。 1. 解压下载的[第三方库](http://pan.baidu.com/s/1jG5o7Am)，里面包含“3rdparty,bin,tools”这三个文件夹，将三个文件夹拷贝到 caffe window 源码目录下，合并。 2. 用 VS 2012 打开解决方案文件 ./build/MSVC/MainBuilder.sln 3. 将编译目标切换到 x64 平台 4. 重新生成解决方案

按照如下步骤，由于 RP 问题，本人并没有直接编译通过，经研究是由于项目中没有配置 CUDA 的环境。将 CUDA 相关配置增加到项目的“包含目录”及“库目录”，项目->属性->VC++目录，编译通过。编译会生成在目录./bin 目录下生成文件 MainCaller.exe， 通过改变 ./examples/MainCaller.cpp 文件内容，选择编译的文件。

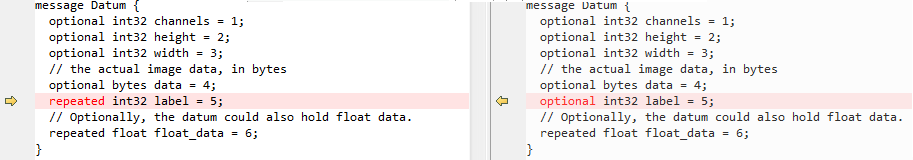
**cafee window 修改（适用于车辆定位问题）**

由于有了之前修改 linux 下的 caffe 源码的相关经验（[博文：Caffe 源码的修改（用于车辆的定位）](http://lxiongh.com/2015/01/20/Caffe_Change_for_Localization/)），这个修改还是得心应手的。 为了将代码修改成适用于多标签的训练与测试，总体来讲，修改了下图中所示的部分，其中左栏为修改后的，右栏为未修改前的。

如下代码变动截图，均利用 Beyond Compare 软件比较得到。

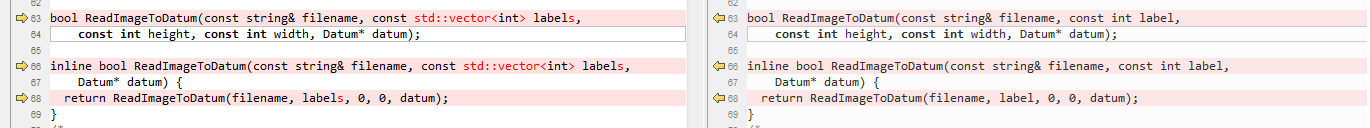


**./src/caffe/proto/caffe.proto**



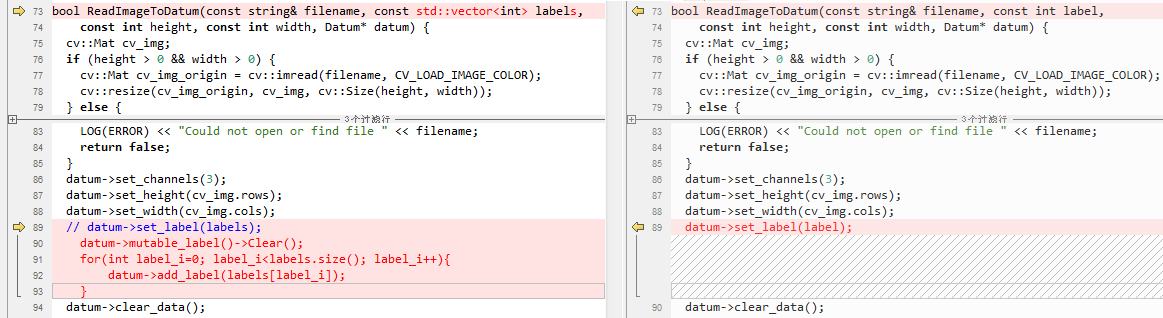
如上图，将 label 由 optional 修改为 repeated，即使其支持多标签输入。 改变这个文件后， 需要用批处理./scripts/GeneratePB.bat 重新生成文件 caffe.pb.cc 与 caffe.pb.h。项目编译前也会自动调用该批处理程序，但如果已经存在文件 caffe.pb.h，则其不会被重新生成。所以在修改了 caffe.proto 后，一定要删除原来的 caffe.pb.cc 与caffe.pb.h 文件，然后再由批处理程序生成，或者在编译前调用生成。

**./include/caffe/util/io.hpp**



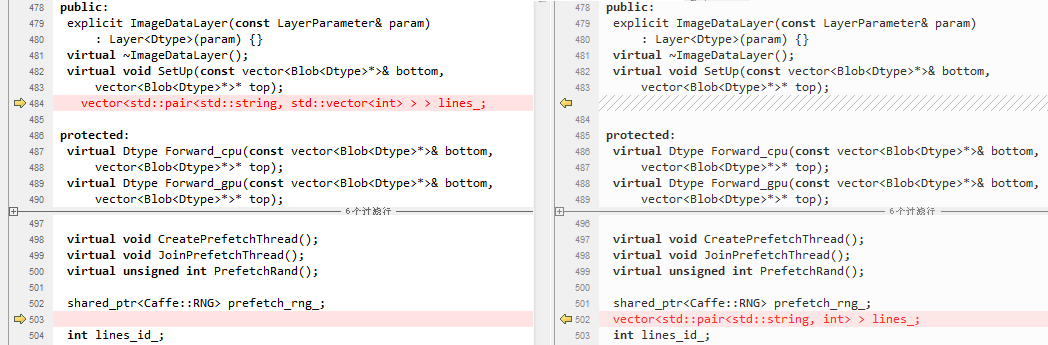
将 label 由 const int 型转变为 const std::vector<int> label， 用于存储多标签信息。同时需要在 io.cpp 文件中作相应的修改。

**./src/caffe/util/io.cpp**



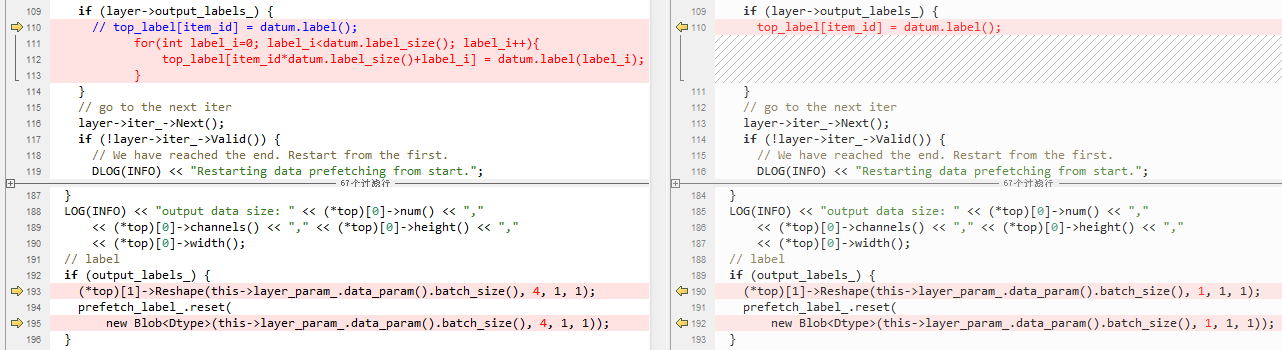
io.cpp 及 io.hpp 是最基本的数据读取实现，在 convert\_imageset.cpp 与 image\_data\_layer.cpp 会直接调用ReadImageToDatum() 函数读取图片数据。

**./include/caffe/vision\_layers.hpp**



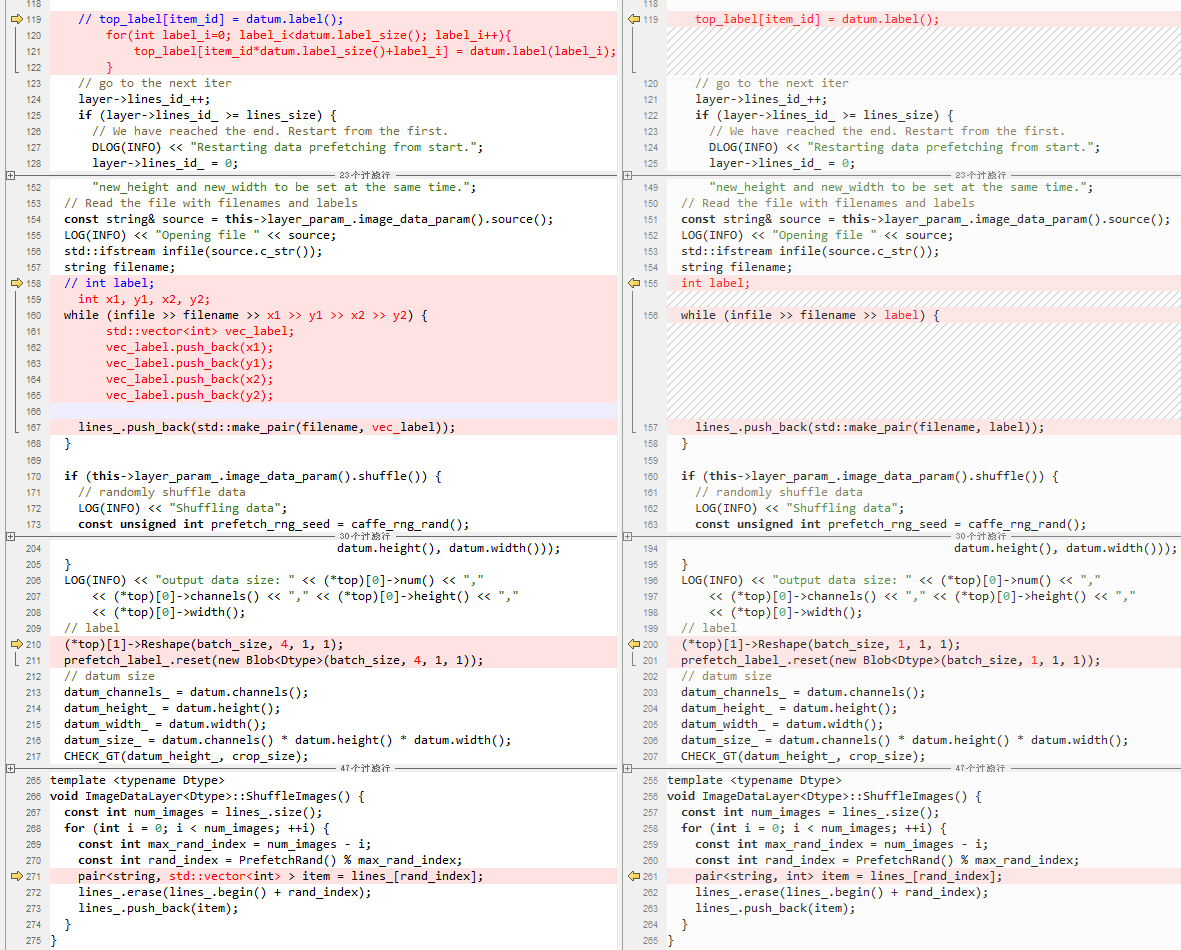
将 lines\_ 由 vector<std::pair<std::string, int> > 转变为 vector<std::pair<std::string, std::vector<int> > >，用于在存储图片的路径及标签信息。特别地，将 lines\_ 由 protected 修饰改为 public 修饰，可只是为了在后续能够更加方便的直接访问图片的路径数据（具体体现在 test\_net.cpp 中）。

**./src/caffe/layers/data\_layer.cpp**



data layer 中数据是从数据库（如 leveldb）中读取中，其并不涉及到图片底层的读取， 因为其数据已经由convert\_imageset.cpp 转化成了 leveldb 形式的数据。 此处只需从 datum 中读取相应的标签数据 datum.label(label\_i)赋值给 top\_label[item\_id\*datum.label\_size()+label\_i] 即可，同时不要忘了分配相应的空间。

**./src/caffe/layers/image\_data\_layer.cpp**



此层直接读取图片数据及标签信息，首先从模型的配置文件 \*.prototxt 中读取 source（里面包含了图片的绝对路径及图片的标签信息），如下所示。

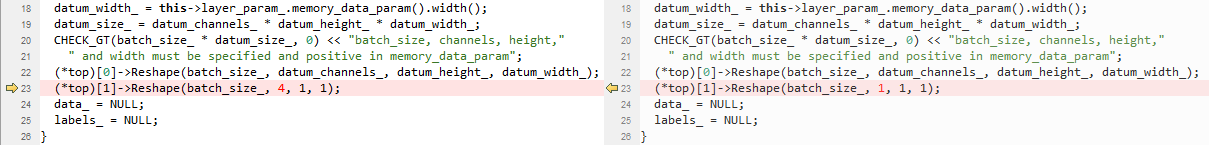
E:/cars/03\_041.jpg 65 85 151 202

E:/cars/03\_042.jpg 84 85 178 201

E:/cars/03\_043.jpg 90 71 183 191

特别地，测试也需给出上例中的文件内容，不同于之处在于用 0 0 0 0 替代相应的标签信息。

**./src/caffe/layers/memory\_data\_layer.cpp**



本实例中没有用到此层，但还是作了相应的修改。

## 车辆检测代码（参考 test\_net.cpp 修改而来）

*// Copyright 2014 BVLC and contributors.*

*//*

*// This is a simple script that allows one to quickly test a network whose*

*// structure is specified by text format protocol buffers, and whose parameter*

*// are loaded from a pre-trained network.*

*// Usage:*

*// test\_net net\_proto pretrained\_net\_proto iterations [CPU/GPU]*

#include <cuda\_runtime.h>

#include <cstring>

#include <cstdlib>

#include <vector>

#include "caffe/caffe.hpp"

*// for read and write image*

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/highgui/highgui\_c.h>

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/opencv.hpp>

#include <fstream>

**using** **namespace** caffe; *// NOLINT(build/namespaces)*

**enum** OUTPUT\_MODE {DRAW, LABEL, BOTH};

int main(int argc, char\*\* argv) {

**if** (argc < 5 || argc > 8) {

LOG(ERROR) << "test\_net net\_proto pretrained\_net\_proto iterations output\_dir"

<< " [DRAW/LABEL/BOTH]"

<< " [CPU/GPU] [Device ID]";

**return** 1;

}

Caffe::set\_phase(Caffe::TEST);

OUTPUT\_MODE mode = BOTH;

**if**(argc >= 6){

**if**(strcmp(argv[5], "DRAW") == 0){

mode = DRAW;

}**else** **if**(strcmp(argv[5], "LABEL") == 0){

mode = LABEL;

}

}

**if** (argc >= 7 && strcmp(argv[6], "GPU") == 0) {

Caffe::set\_mode(Caffe::GPU);

int device\_id = 0;

**if** (argc == 8) {

device\_id = atoi(argv[7]);

}

Caffe::SetDevice(device\_id);

LOG(ERROR) << "Using GPU #" << device\_id;

} **else** {

LOG(ERROR) << "Using CPU";

Caffe::set\_mode(Caffe::CPU);

}

Net<float> caffe\_test\_net(argv[1]);

caffe\_test\_net.CopyTrainedLayersFrom(argv[2]);

int total\_iter = atoi(argv[3]);

string output\_dir(argv[4]);

*//creat output\_dir*

string mkdir\_cmd("mkdir " + output\_dir);

int found = mkdir\_cmd.find("/");

**while**(found > 0){*// replace invalid "/" with "\" supported by system command*

mkdir\_cmd.replace(found, 1, "**\\**");

found = mkdir\_cmd.find("/");

}

system(mkdir\_cmd.c\_str());

LOG(ERROR) << "Running " << total\_iter << " iterations.";

vector<Blob<float>\*> input\_vec;

int image\_index = 0;

**const** vector<shared\_ptr<Layer<float> > > layers = caffe\_test\_net.layers();

ImageDataLayer<float>\* image\_layer = **reinterpret\_cast**<ImageDataLayer<float>\*>(layers[0].get());

CHECK(image\_layer);

int new\_height = image\_layer->layer\_param().image\_data\_param().new\_height();

int new\_width = image\_layer->layer\_param().image\_data\_param().new\_width();

**for** (int i = 0; i < total\_iter; ++i) {

caffe\_test\_net.Forward(input\_vec);

**const** shared\_ptr<Blob<float> > blob\_ = caffe\_test\_net.blob\_by\_name("fc\_8\_det");

**const** float\* det\_points = blob\_->mutable\_cpu\_data();

**for**(int n=0; n<blob\_->num(); n++){

det\_points = blob\_->mutable\_cpu\_data() + blob\_->offset(n);

string image\_path(image\_layer->lines\_[image\_index].first);

found = image\_path.find\_last\_of("/**\\**");

string fn(image\_path.substr(found+1));

*//LOG(ERROR) << output\_dir << "/" << fn << " " \*

*<< det\_points[0] << " " << det\_points[1] << " " << det\_points[2] << " " << det\_points[3] << "\n";*

int x1 = det\_points[0];

int y1 = det\_points[1];

int x2 = det\_points[2];

int y2 = det\_points[3];

cv::Mat cv\_img;

cv\_img = cv::imread(image\_path, CV\_LOAD\_IMAGE\_COLOR);

CHECK(!cv\_img.empty());

**if**(new\_height > 0 && new\_width > 0){

*// rescale points*

x1 = float(x1)/new\_width \* cv\_img.cols;

y1 = float(y1)/new\_height \* cv\_img.rows;

x2 = float(x2)/new\_width \* cv\_img.cols;

y2 = float(y2)/new\_height \* cv\_img.rows;

}

*// draw boundary in image*

**if**(mode==BOTH || mode==DRAW){

*// write image with boundary*

line(cv\_img, cv::Point(x1, y1), cv::Point(x2, y1), cv::Scalar(0, 0, 255), 3);

line(cv\_img, cv::Point(x2, y1), cv::Point(x2, y2), cv::Scalar(0, 0, 255), 3);

line(cv\_img, cv::Point(x2, y2), cv::Point(x1, y2), cv::Scalar(0, 0, 255), 3);

line(cv\_img, cv::Point(x1, y2), cv::Point(x1, y1), cv::Scalar(0, 0, 255), 3);

*// write image*

string out\_path(output\_dir+"/"+fn);

CHECK\_EQ(imwrite(out\_path, cv\_img), true) << "write image " + out\_path + " failed";

}

**if**(mode==BOTH || mode==LABEL){

*// write image predict label*

std::fstream fs;

found = fn.find(".");

string fpath(output\_dir+"/"+fn.substr(0,found)+".lbl");

fs.open(fpath, std::ios::out);

**if**(fs.is\_open()){

fs << x1 << " " << y1 << " " << x2 << " " << y2;

fs.close();

}**else**{

LOG(ERROR) << "Can't create " << fpath << "**\n**";

}

}

LOG(ERROR) << "[" << image\_index+1 << "/" << image\_layer->lines\_.size() << "]" << " " << fn << "**\n**";

image\_index ++;

**if**(image\_index >= image\_layer->lines\_.size()){

LOG(ERROR) << "restart from begining**\n**";

image\_index = 0;

}

}

}

**return** 0;

}

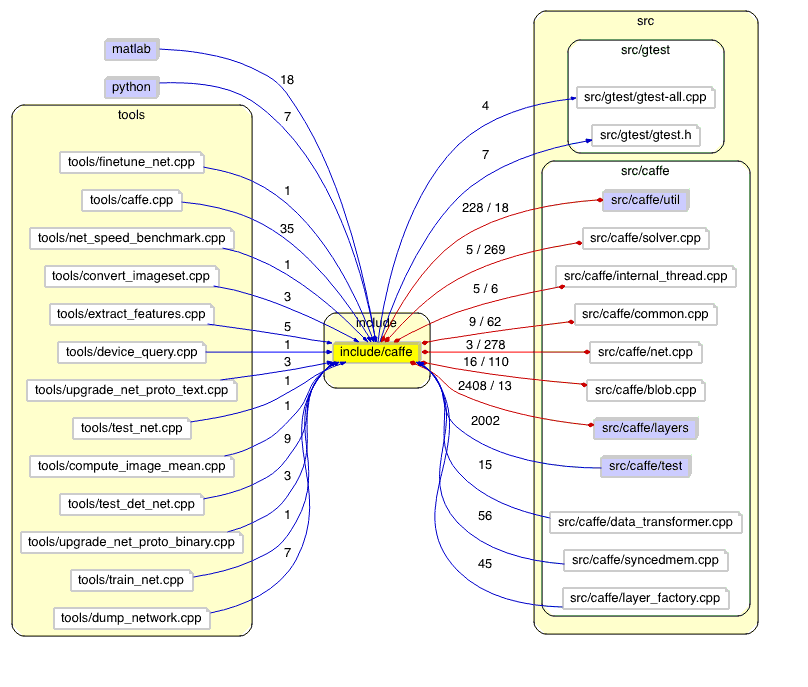
## Caffe 源码的修改（用于车辆的定位）

[主页](http://lxiongh.com/)[分类](http://lxiongh.com/categories/)[标签](http://lxiongh.com/tags/)[链接](http://lxiongh.com/links/)[关于](http://lxiongh.com/about/)

修改后的caffe源码 <https://gitcafe.com/lxiongh/Caffe_for_Multi-label>

Caffe 的源码仅适用于模式分类问题，其标签是一维的。为了能够将 Caffe 代码应用于车辆的检测问题上，首先要解决的问题就是将 Caffe 源码中与标签相关的代码由一维改为多维。本项目由于只需预测车辆的上下两角的坐标（x1,y1, x2,y2），即将标签修改为4维。

这次能够很顺利的修改源码，很大一原因是实验室的师弟太给力了，他们对 Caffe 源码的研究开始地很早，并在以 Caffe 作为其深度学习的工具，并且在也发表了一些很不错的文章。Thanks to TianShui, LiangLi, JinZhu. 『车辆定位项目链接 <http://lxiongh.github.io/2015/02/04/car_localization/> 』

首先，利用 Understanding 软件，可以方便的查看到 caffe 源码的目录结构，如下图所示。

可以注意到，在 Caffe 源码里有一个『Tools』的目录，里面有一些相当有用的工具，如『compute\_image\_mean.cpp』、『convert\_imageset.cpp』等，其中『convert\_imageset.cpp』直接操作到了文本文件，如下列代码所示。那么函数『ReadImageToDatum』将就是突破口。

1 *// convert\_imageset.cpp, line 129-*

2 **for** (int line\_id = 0; line\_id < lines.size(); ++line\_id) {

3 **if** (!ReadImageToDatum(root\_folder + lines[line\_id].first,

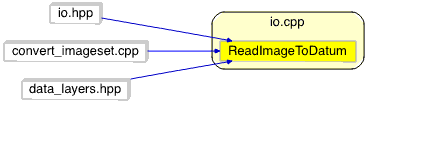
4 lines[line\_id].second, resize\_height, resize\_width, is\_color, &datum)) {

5 **continue**;

6 }

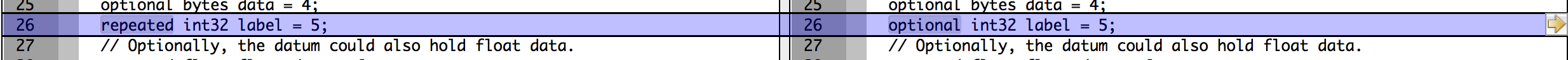
7 *// ...*

8 }



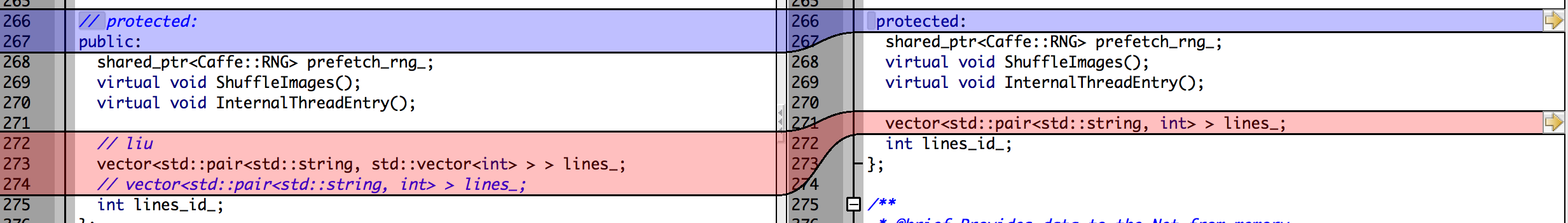
定义在io.cpp里的函数ReadImageToDatum完成了将图片数据转换成caffe能够处理的Datum类型，主要修改的文件大都集中在数据层，将其单标签改成多标签支持。针对车辆检测，对caffe所作的修改有如下的部分：

* caffe.proto



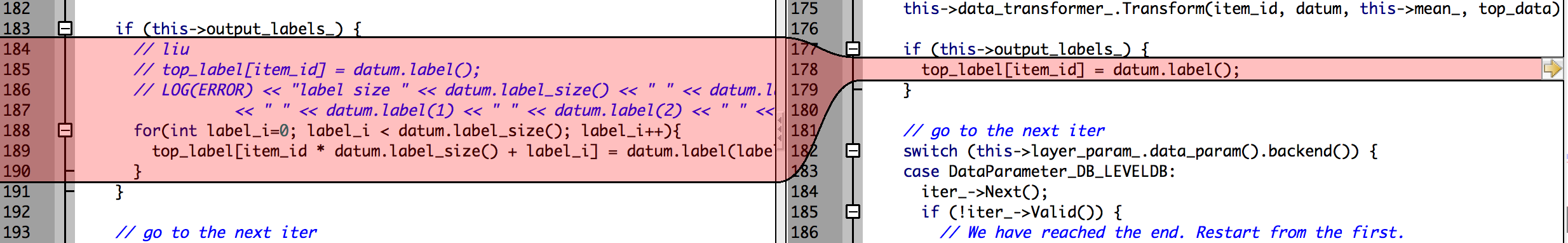
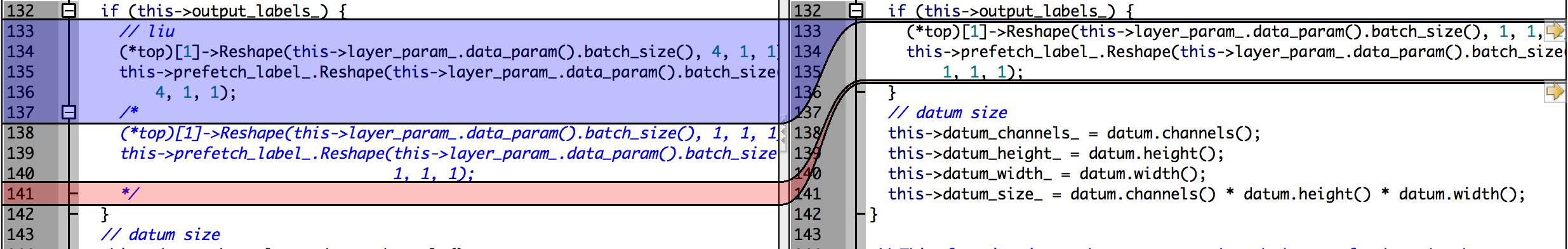
将optional改为repeated，使得标签变量label为数组，即支持多标签。否则无此属性Datum.label\_size()。

* data\_layer.hpp



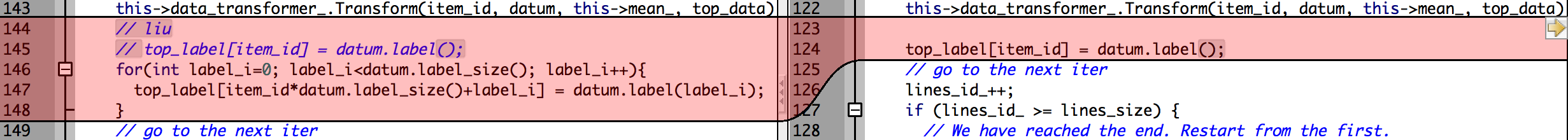
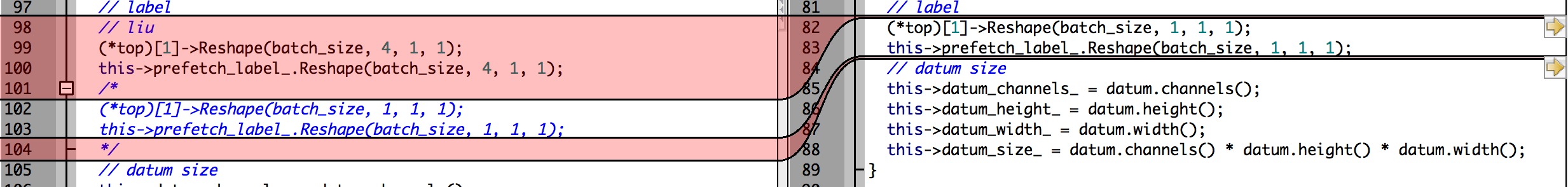
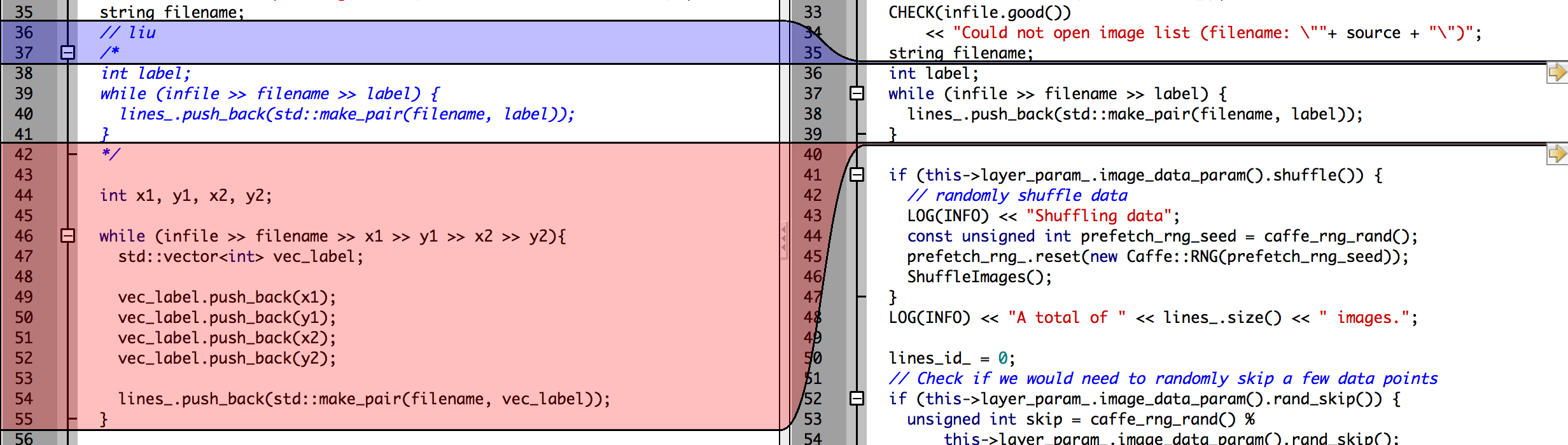
将lines\_由protected修改为public，使得后续能够利用指针直接访问lines\_数据，其中保存了图片名及其对应的标签信息。详见test\_det\_net.cpp

* data\_layer.cpp



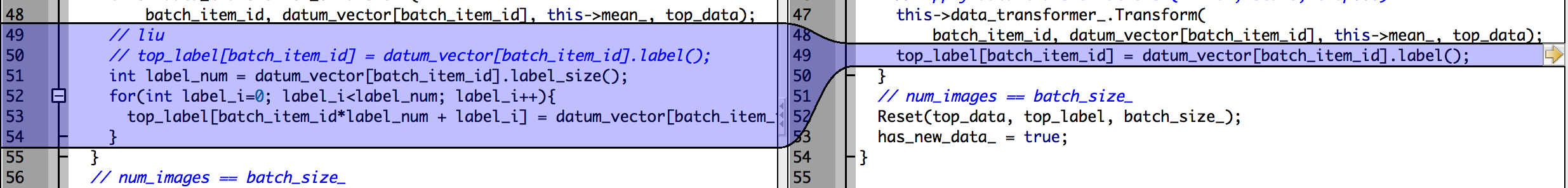
修改top\_label，使得其保存图片的多标签信息。

* image\_data\_layer.cpp

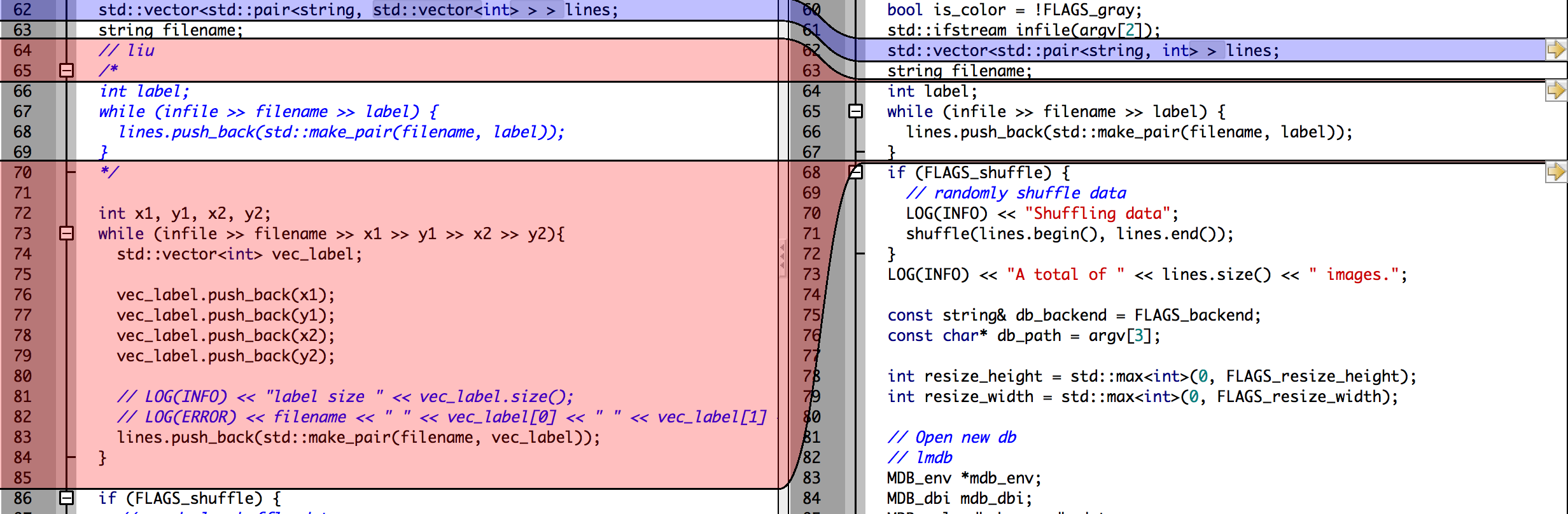


从文本文件里读取图片的路径及标签信息，将原来int label修改成std::<vector> vec\_label。同时需要特别注意的就是不要忘记申请相应的存储空间(\*top)[1]->Reshape(this->...)，否则在初始化网络时就会出现错误。

* memory\_data\_layer.cpp

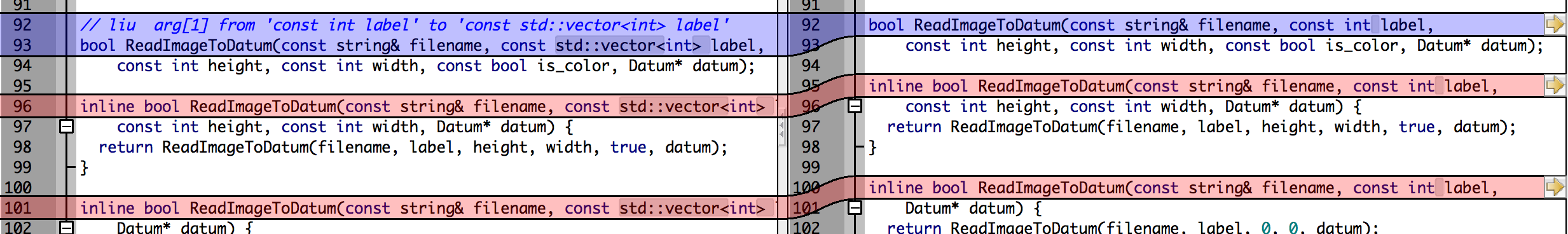


虽然这个在实际应用中没有用到，但因其涉及到最底层的数据层，所以也修改了。

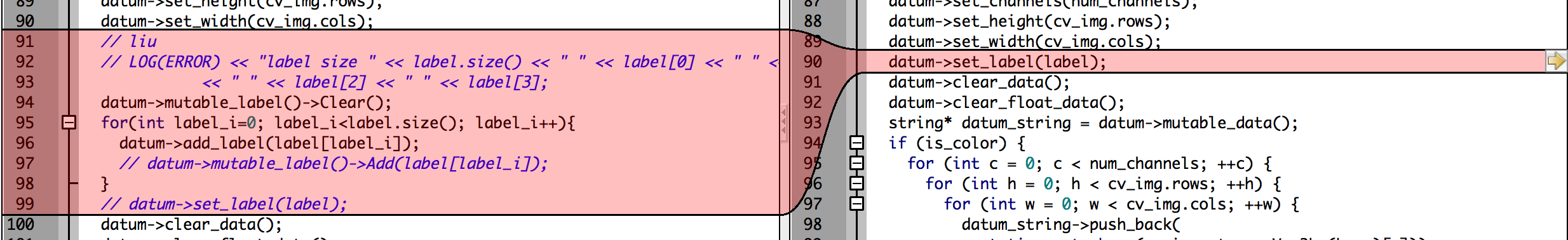
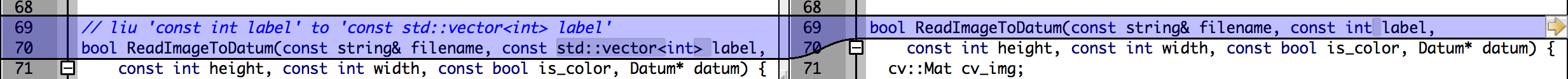
* convert\_imageset.cpp

这个程序是将图片数据打包与数据库的形式，默认为leveldb

* io.hpp



* io.cpp



这个文件涉及到最底层的数据读写工作。

* test\_det\_net.cpp，由extract\_feature.cpp修改而来的对输入的图片进行车辆预测，并画框输出

1 #include <stdio.h> *// for snprintf*

2 #include <string>

3 #include <vector>

4

5 #include "boost/algorithm/string.hpp"

6 #include "google/protobuf/text\_format.h"

7 #include "leveldb/db.h"

8 #include "leveldb/write\_batch.h"

9

10 #include "caffe/blob.hpp"

11 #include "caffe/common.hpp"

12 #include "caffe/net.hpp"

13 #include "caffe/proto/caffe.pb.h"

14 #include "caffe/util/io.hpp"

15 #include "caffe/vision\_layers.hpp"

16

17 *// liu*

18 #include <opencv2/core/core.hpp>

19 #include <opencv2/highgui/highgui.hpp>

20 #include <opencv2/highgui/highgui\_c.h>

21 #include <opencv2/imgproc/imgproc.hpp>

22 #include <opencv2/opencv.hpp>

23 #include <sys/stat.h> *// for mkdir*

24

25 **using** **namespace** caffe; *// NOLINT(build/namespaces)*

26

27 **template**<**typename** Dtype>

28 int feature\_extraction\_pipeline(int argc, char\*\* argv);

29

30 int main(int argc, char\*\* argv) {

31 **return** feature\_extraction\_pipeline<float>(argc, argv);

32 *// return feature\_extraction\_pipeline<double>(argc, argv);*

33 }

34

35 **template**<**typename** Dtype>

36 int feature\_extraction\_pipeline(int argc, char\*\* argv) {

37 ::google::InitGoogleLogging(argv[0]);

38 **const** int num\_required\_args = 5;

39 **if** (argc < num\_required\_args) {

40 LOG(ERROR)<<

41 "This program takes in a trained network and an input data layer, and then"

42 " extract features of the input data produced by the net.**\n**"

43 "Usage: test\_det\_net pretrained\_net\_param"

44 " feature\_extraction\_proto\_file num\_mini\_batches"

45 " output\_dir"

46 " [CPU/GPU] [DEVICE\_ID=0]**\n**"

47 "Note: the feature blob names is fixed as 'fc\_8\_det' in code**\n**";

48 **return** 1;

49 }

50 int arg\_pos = num\_required\_args;

51

52 arg\_pos = num\_required\_args;

53 **if** (argc > arg\_pos && strcmp(argv[arg\_pos], "GPU") == 0) {

54 LOG(ERROR)<< "Using GPU";

55 uint device\_id = 0;

56 **if** (argc > arg\_pos + 1) {

57 device\_id = atoi(argv[arg\_pos + 1]);

58 CHECK\_GE(device\_id, 0);

59 }

60 LOG(ERROR) << "Using Device\_id=" << device\_id;

61 Caffe::SetDevice(device\_id);

62 Caffe::set\_mode(Caffe::GPU);

63 } **else** {

64 LOG(ERROR) << "Using CPU";

65 Caffe::set\_mode(Caffe::CPU);

66 }

67 Caffe::set\_phase(Caffe::TEST);

68

69 arg\_pos = 0; *// the name of the executable*

70 string pretrained\_binary\_proto(argv[++arg\_pos]);

71

72 string feature\_extraction\_proto(argv[++arg\_pos]);

73 shared\_ptr<Net<Dtype> > feature\_extraction\_net(

74 **new** Net<Dtype>(feature\_extraction\_proto));

75 feature\_extraction\_net->CopyTrainedLayersFrom(pretrained\_binary\_proto);

76 *// to get image\_paths*

77 **const** vector<shared\_ptr<Layer<float> > > layers = feature\_extraction\_net->layers();

78 **const** caffe::ImageDataLayer<float> \*image\_layer = **dynamic\_cast**<caffe::ImageDataLayer<float>\* >(layers[0].get());

79 CHECK(image\_layer);

80

81 **const** string blob\_name = "fc\_8\_det";

82

83 CHECK(feature\_extraction\_net->has\_blob(blob\_name)) \

84 << "Unknown feature blob name " << blob\_name \

85 << " in the network " << feature\_extraction\_proto;

86

87

88 int num\_mini\_batches = atoi(argv[++arg\_pos]);

89 string output\_dir = argv[++arg\_pos];

90 CHECK\_EQ(mkdir(output\_dir.c\_str(),0744), 0) << "mkdir " << output\_dir << " failed";

91

92 LOG(ERROR)<< "Extracting Features";

93

94 vector<Blob<float>\*> input\_vec;

95 int image\_index=0;

96 **for** (int batch\_index = 0; batch\_index < num\_mini\_batches; ++batch\_index) {

97 feature\_extraction\_net->Forward(input\_vec);

98

99 **const** shared\_ptr<Blob<Dtype> > feature\_blob = feature\_extraction\_net->blob\_by\_name(blob\_name);

100

101 int batch\_size = feature\_blob->num();

102

103 int dim\_features = feature\_blob->count() / batch\_size;

104 CHECK\_EQ(dim\_features, 4) << "the dim of feature is not equal to 4";

105

106 Dtype\* feature\_blob\_data;

107 int x1, y1, x2, y2;

108 **for** (int n = 0; n < batch\_size; ++n) {

109 feature\_blob\_data = feature\_blob->mutable\_cpu\_data() + feature\_blob->offset(n);

110

111 x1 = feature\_blob\_data[0];

112 y1 = feature\_blob\_data[1];

113 x2 = feature\_blob\_data[2];

114 y2 = feature\_blob\_data[3];

115

116 string image\_path = image\_layer->lines\_[image\_index].first;

117 *//LOG(ERROR) << "image\_index " << image\_index << " " << image\_path \*

118  *<< " x1 " << feature\_blob\_data[0] << " y1 " << feature\_blob\_data[1] \*

119  *<< " x2 " << feature\_blob\_data[2] << " y2 " << feature\_blob\_data[3];*

120

121 cv::Mat img\_origin = cv::imread(image\_path);

122

123 std::vector<string> part\_names;

124 boost::split(part\_names, image\_path, boost::is\_any\_of("/"));

125 string subname = part\_names[part\_names.size()-1]; *// the last element is the image name.*

126 string out\_path(output\_dir + "/" + subname);

127

128 *//LOG(ERROR) << subname;*

129 line(img\_origin, cv::Point(x1, y1), cv::Point(x2, y1), cv::Scalar(0, 0, 255), 3);

130 line(img\_origin, cv::Point(x2, y1), cv::Point(x2, y2), cv::Scalar(0, 0, 255), 3);

131 line(img\_origin, cv::Point(x2, y2), cv::Point(x1, y2), cv::Scalar(0, 0, 255), 3);

132 line(img\_origin, cv::Point(x1, y2), cv::Point(x1, y1), cv::Scalar(0, 0, 255), 3);

133 CHECK\_EQ(imwrite(output\_dir + "/" + subname, img\_origin), true) << "write image " + out\_path + " failed";

134

135 image\_index ++ ;

136 **if** (image\_index>=image\_layer->lines\_.size()){

137 LOG(ERROR) << "Restarting data prefetching from start.";

138 image\_index = 0;

139 }

140 *// (image\_index>image\_layer->lines\_.size()-1)?(image\_index=0):(image\_index++);*

141 }

142

143 } *// for (int batch\_index = 0; batch\_index < num\_mini\_batches; ++batch\_index)*

144 *// write the last batch*

145

146 LOG(ERROR)<< "Successfully extracted the features!";

147 **return** 0;

148 }

# Deep Localization Network for car

[主页](http://lxiongh.com/)[分类](http://lxiongh.com/categories/)[标签](http://lxiongh.com/tags/)[链接](http://lxiongh.com/links/)[关于](http://lxiongh.com/about/)

利用了 X.L. Wang [NPIP 2014] (Deep Joint Task Learning for Generic Object Extraction)中的 Localization Network,输入为 227\*227 的 RGB 图片,网 络输出为上下角的坐标(x1,y1,x2,y2)。在 X.L. Wang [NIP 2014]的 Localization 网络模型(从 Imagenet 训练得到)的基础上, 利用 2500 张标定好的车辆数据进行 fine-tuning。其中 2000 张用于训练,剩下 500 张用于测试。 本项目源码修改可移步 <http://lxiongh.com/2015/01/20/Caffe_Change_for_Localization/>

## Localization Network and Some Results

