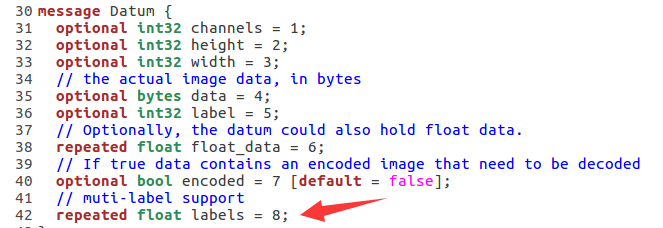
# 修改caffe源码实现MDB格式多标签回归

由于Caffe中使用LMDB格式只支持单标签，而HDF5格式虽然支持多标签，但体积太大，而使用LMDB格式，体积小，且可以在网络文件中进行很多的数据增强操作，如减裁、镜向、加入均值文件等。这里修改Caffe的源码，使LMDB格式支持多标签，结合人脸坐标回归任务，说明该方法。

## 修改与数据转换相关的源码

### 修改caffe.proto文件

在caffe/src/caffe/proto目录下，打开caffe.proto文件，在message Datum下增加一个repeated float labels以支持多标签：



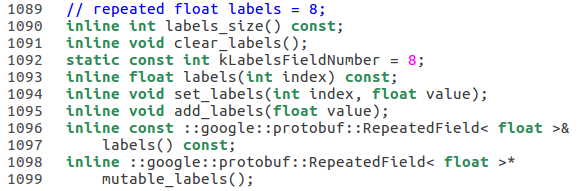
在message DataParameter中增加一个optional uint 32 label\_number用来指定标签的数量。



将原来的caffe/build文件夹中的内容删除，重新编译caffe，在caffe/build/include/caffe/proto文件夹下，生成如下文件：

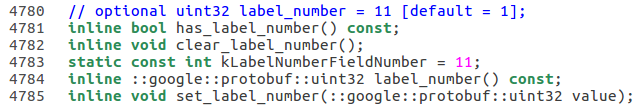


打开caffe.pb.h文件，查找到如下内容：



这里在caffe为我们生成了add\_labels和labels等函数，其中add\_labels用于添加从此标签，labels且于返回指定序号的标签。

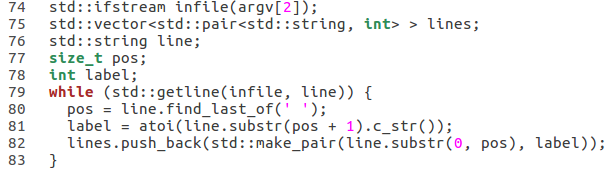
同样的，可以找到以下代码：



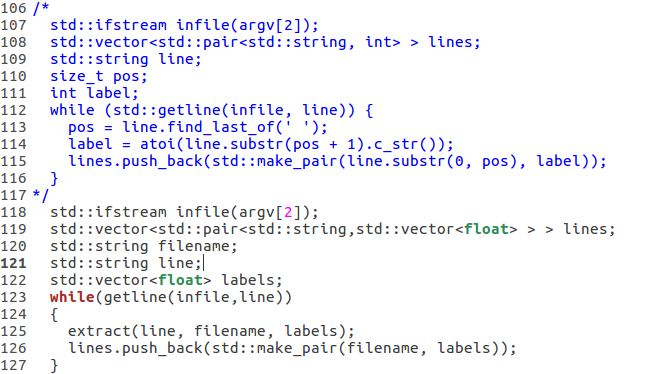
这些函数用来操作label\_number。

### 编写数据转换文件

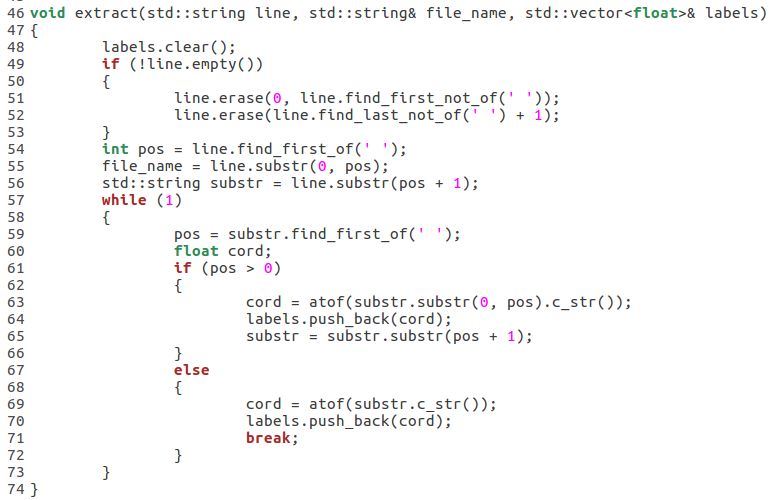
在caffe中，将图像数据转换成LMDB格式是使用convert\_imageset工具来实现的，该工具源码在caffe/tools文件夹下，这里在复制一份convert\_imageset.cpp在同一文件夹下，更名为create\_multilabel\_lmdb.cpp，打开文件，查找到如下代码：



这里从文件中循环读取行，然后将行文本按空格分割，然后将图像文件名及对应的标签保存到vector类型的lines中。这里是针对单标签的，为了支持多标签，我们需要定义一个vector<float> 类型向量来保存多个标签，这里在以4个标签为例，把图像文件名及对应的标签保存到vector<pair<string,vector<float>>> 类型的向量中，因此，将以上代码注释掉，添加如下代码：



其中extract函数定义如下：



该函数用于从行字符串中分离出图像名称和对应的标签。

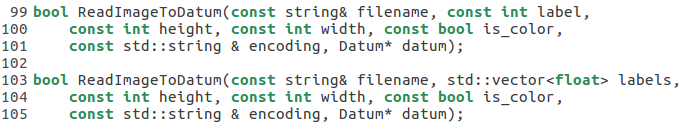
在原文件中，找到如下代码：



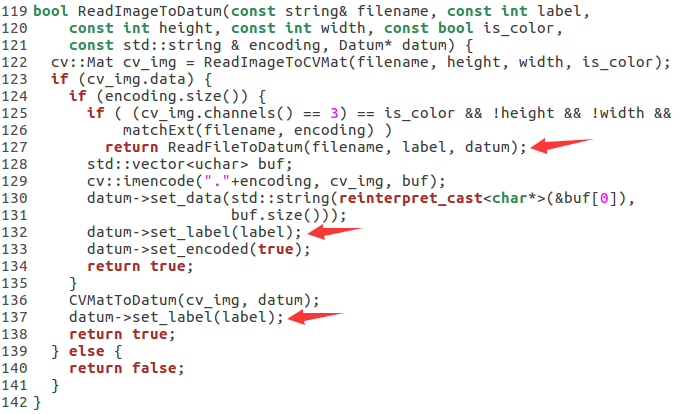
这里使用ReadImageToDatum函数读取图像和对应的标签保存到Datum中，该函数的定义和实现在caffe/include/caffe/util/io.hpp和caffe/src/caffe/util/io.cpp中，我们重载该函数，使其能够处理多标签。

### io.hpp和io.cpp文件的改写

在io.hpp中：



第一个ReadImageToDatum为原函数，第二个ReadImageToDatum为我们重载的函数。我们在io.cpp参考原函数的实现来实现它，先看原函数的实现：



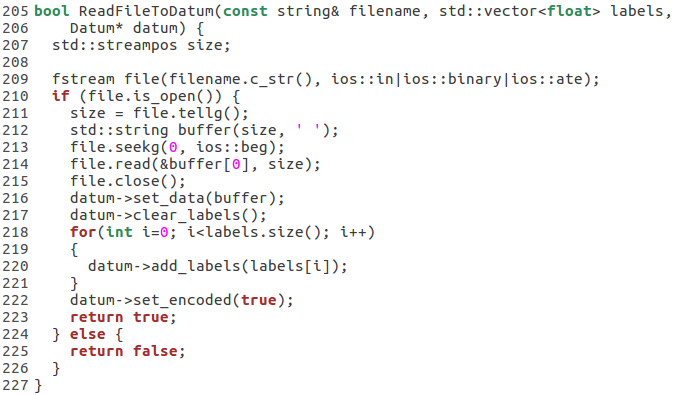
凡是对单标签label进行处理的地方，我们都要改写。

1. 第一处，同样我们要重载ReadFileToDatum函数。

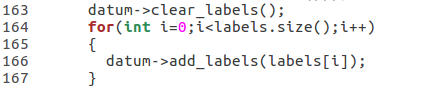
首先在io.hpp中声明：



在io.cpp中实现：



1. 第二处，set\_label是针对单标签的，这里们需要使用支持多标签add\_labels，修改如下：



1. 第三处，同第二处。

为方便观察读取的标签是否正确，我们在函数开始时打印出读取到的文件名和标签信息，重载的ReadImageToDatum函数全部代码如下：



## 生成多标签LMDB文件

经过以上的修改，重新编译caffe，此时在build/tools文件夹下生成create\_mutilabel\_lmdb，我们可以创建多标签的LMDB格式文件了。

### 数据准备

在caffe/examples下新建一个multi\_labels\_test文件夹，下面包括三个文件：



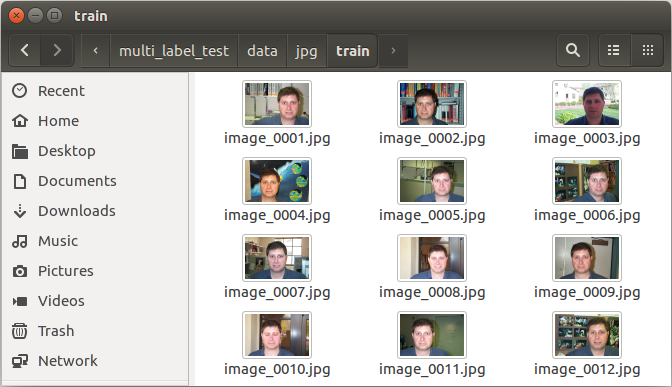
其中data中包括以下文件夹：

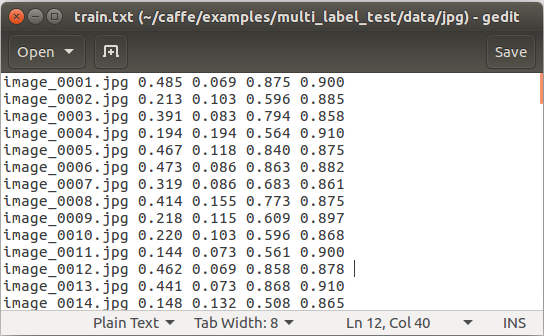


jpg用于存放原始的图像文件，lmdb用于存放转换后的lmdb文件。jpg文件内容如下：



train和val下为原始图像，train.txt和val.txt为这些图像文件名及对应的标签，其中train.txt中的格式如图所示。





使用的图像大小为896\*592，标签值为归一化的人脸坐标。

### 生成lmdb文件

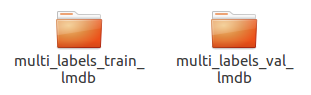
在multi\_label\_test/net下新建脚本文件create\_multi-label\_lmdb.sh，文件内容如下：



在caffe目录下执行脚本：

./examples/multi\_label\_test/net/create\_multi-label\_lmdb.sh

在caffe/examples/multi\_label\_test/data/lmdb下生成以下文件：



同时在屏幕上输出处理的图像文件名及标签信息。

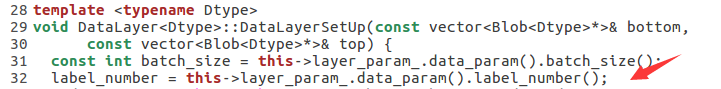
## 修改与训练相关的源码

生成多标签的LMDB文件后，需要修改data\_layer.hpp和data\_layer.cpp，以适应该文件的输入。

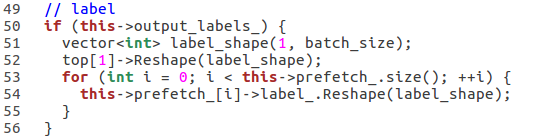
Data\_layer.hpp在caffe/include/caffe/layers目录下，打开为Datalayer类增加一个int型label\_number变量。



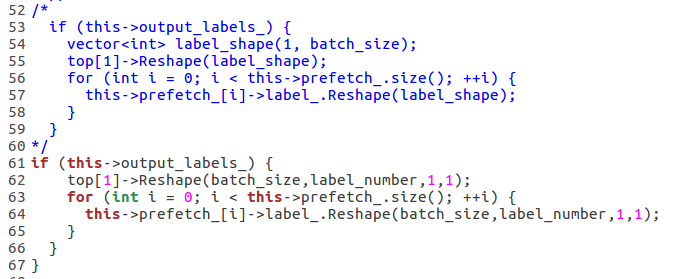
data\_layer.cpp在caffe/src/caffe/layers目录下，打开在DataLayerSetUp函数中读取label\_number的值。



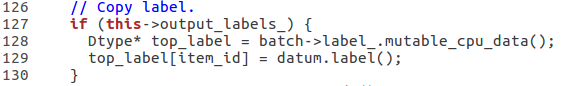
找到如下代码：



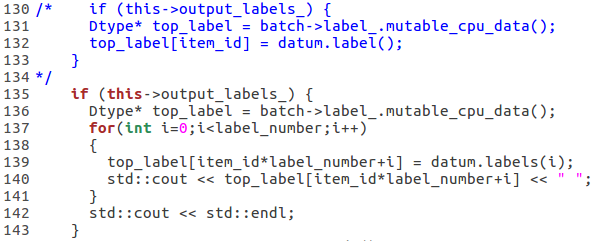
这里为labels分配空间，修改成：



另外，找到以下代码：



此处把datum中的的标签拷贝到top\_label数组中，修改成：



修改后，重新编译caffe。

## 训练

### train\_val.prototxt

在该文件中，需要为我们之前增加的label\_number赋值，这里每一张图像有4个标签，所以label\_number的值为4。

name: "CaffeNet"

layer {

name: "data"

type: "Data"

top: "data"

top: "label"

include {

phase: TRAIN

}

# transform\_param {

# mirror: true

# crop\_size: 227

# mean\_file: "data/ilsvrc12/imagenet\_mean.binaryproto"

# }

# mean pixel / channel-wise mean instead of mean image

transform\_param {

crop\_size: 227

mean\_value: 104

mean\_value: 117

mean\_value: 123

mirror: true

}

data\_param {

source: "/home/hy17003/caffe/examples/multi\_label\_test/data/lmdb/multi\_labels\_train\_lmdb"

batch\_size: 100

**label\_number: 4**

backend: LMDB

}

}

layer {

name: "data"

type: "Data"

top: "data"

top: "label"

include {

phase: TEST

}

# transform\_param {

# mirror: false

# crop\_size: 227

# mean\_file: "data/ilsvrc12/imagenet\_mean.binaryproto"

# }

# mean pixel / channel-wise mean instead of mean image

transform\_param {

crop\_size: 227

mean\_value: 104

mean\_value: 117

mean\_value: 123

mirror: false

}

data\_param {

source: "/home/hy17003/caffe/examples/multi\_label\_test/data/lmdb/multi\_labels\_val\_lmdb"

batch\_size: 100

**label\_number: 4**

backend: LMDB

}

}

layer {

name: "conv1"

type: "Convolution"

bottom: "data"

top: "conv1"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 96

kernel\_size: 11

stride: 4

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

name: "relu1"

type: "ReLU"

bottom: "conv1"

top: "conv1"

}

layer {

name: "pool1"

type: "Pooling"

bottom: "conv1"

top: "pool1"

pooling\_param {

pool: MAX

kernel\_size: 3

stride: 2

}

}

layer {

name: "norm1"

type: "LRN"

bottom: "pool1"

top: "norm1"

lrn\_param {

local\_size: 5

alpha: 0.0001

beta: 0.75

}

}

layer {

name: "conv2"

type: "Convolution"

bottom: "norm1"

top: "conv2"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 256

pad: 2

kernel\_size: 5

group: 2

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 1

}

}

}

layer {

name: "relu2"

type: "ReLU"

bottom: "conv2"

top: "conv2"

}

layer {

name: "pool2"

type: "Pooling"

bottom: "conv2"

top: "pool2"

pooling\_param {

pool: MAX

kernel\_size: 3

stride: 2

}

}

layer {

name: "norm2"

type: "LRN"

bottom: "pool2"

top: "norm2"

lrn\_param {

local\_size: 5

alpha: 0.0001

beta: 0.75

}

}

layer {

name: "conv3"

type: "Convolution"

bottom: "norm2"

top: "conv3"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 384

pad: 1

kernel\_size: 3

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

name: "relu3"

type: "ReLU"

bottom: "conv3"

top: "conv3"

}

layer {

name: "conv4"

type: "Convolution"

bottom: "conv3"

top: "conv4"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 384

pad: 1

kernel\_size: 3

group: 2

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 1

}

}

}

layer {

name: "relu4"

type: "ReLU"

bottom: "conv4"

top: "conv4"

}

layer {

name: "conv5"

type: "Convolution"

bottom: "conv4"

top: "conv5"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 256

pad: 1

kernel\_size: 3

group: 2

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 1

}

}

}

layer {

name: "relu5"

type: "ReLU"

bottom: "conv5"

top: "conv5"

}

layer {

name: "pool5"

type: "Pooling"

bottom: "conv5"

top: "pool5"

pooling\_param {

pool: MAX

kernel\_size: 3

stride: 2

}

}

layer {

name: "fc6"

type: "InnerProduct"

bottom: "pool5"

top: "fc6"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

inner\_product\_param {

num\_output: 4096

weight\_filler {

type: "gaussian"

std: 0.005

}

bias\_filler {

type: "constant"

value: 1

}

}

}

layer {

name: "relu6"

type: "ReLU"

bottom: "fc6"

top: "fc6"

}

layer {

name: "drop6"

type: "Dropout"

bottom: "fc6"

top: "fc6"

dropout\_param {

dropout\_ratio: 0.5

}

}

layer {

name: "fc7"

type: "InnerProduct"

bottom: "fc6"

top: "fc7"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

inner\_product\_param {

num\_output: 4096

weight\_filler {

type: "gaussian"

std: 0.005

}

bias\_filler {

type: "constant"

value: 1

}

}

}

layer {

name: "relu7"

type: "ReLU"

bottom: "fc7"

top: "fc7"

}

layer {

name: "drop7"

type: "Dropout"

bottom: "fc7"

top: "fc7"

dropout\_param {

dropout\_ratio: 0.5

}

}

layer {

name: "fc8"

type: "InnerProduct"

bottom: "fc7"

top: "fc8"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

inner\_product\_param {

num\_output: 4

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

name: "loss"

type: "EuclideanLoss"

bottom: "fc8"

bottom: "label"

top: "loss"

}

### solver.prototxt

net: "/home/hy17003/caffe/examples/multi\_label\_test/net/train\_val.prototxt"

test\_iter: 100test\_interval: 100base\_lr: 0.001lr\_policy: "step"gamma: 0.1stepsize: 10000display: 20max\_iter: 20000momentum: 0.9weight\_decay: 0.0005snapshot: 10000snapshot\_prefix: "/home/hy17003/caffe/examples/multi\_label\_test/result/"solver\_mode: CPU

### 训练脚本文件

#!/usr/bin/env sh

set -e

build/tools/caffe train \

--solver= examples/multi\_label\_test/net/solver.prototxt $@

### 训练

在caffe下执行

./examples/multi\_label\_test/net/train\_net.sh >& train\_log.log

训练日志：略

## 测试

测试使用大小为896\*592的图像，由于网络输入为227\*227，因此要在输入网络之前将其转为227\*227，这里结合opencv的dnn模块进行测试，需要准备以下文件：图像文件，测试网络文件deploy.prototxt，代码文件和训练结果文件，下面贴出deploy.prototxt和代码文件detect.cpp文件代码。

### deploy.prototxt

name: "CaffeNet"

input: "data"

input\_shape {

dim: 1 # batchsize

dim: 1 # number of colour channels - rgb

dim: 227 # width

dim: 227 # height

}

layer {

name: "conv1"

type: "Convolution"

bottom: "data"

top: "conv1"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 96

kernel\_size: 11

stride: 4

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

name: "relu1"

type: "ReLU"

bottom: "conv1"

top: "conv1"

}

layer {

name: "pool1"

type: "Pooling"

bottom: "conv1"

top: "pool1"

pooling\_param {

pool: MAX

kernel\_size: 3

stride: 2

}

}

layer {

name: "norm1"

type: "LRN"

bottom: "pool1"

top: "norm1"

lrn\_param {

local\_size: 5

alpha: 0.0001

beta: 0.75

}

}

layer {

name: "conv2"

type: "Convolution"

bottom: "norm1"

top: "conv2"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 256

pad: 2

kernel\_size: 5

group: 2

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 1

}

}

}

layer {

name: "relu2"

type: "ReLU"

bottom: "conv2"

top: "conv2"

}

layer {

name: "pool2"

type: "Pooling"

bottom: "conv2"

top: "pool2"

pooling\_param {

pool: MAX

kernel\_size: 3

stride: 2

}

}

layer {

name: "norm2"

type: "LRN"

bottom: "pool2"

top: "norm2"

lrn\_param {

local\_size: 5

alpha: 0.0001

beta: 0.75

}

}

layer {

name: "conv3"

type: "Convolution"

bottom: "norm2"

top: "conv3"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 384

pad: 1

kernel\_size: 3

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

name: "relu3"

type: "ReLU"

bottom: "conv3"

top: "conv3"

}

layer {

name: "conv4"

type: "Convolution"

bottom: "conv3"

top: "conv4"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 384

pad: 1

kernel\_size: 3

group: 2

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 1

}

}

}

layer {

name: "relu4"

type: "ReLU"

bottom: "conv4"

top: "conv4"

}

layer {

name: "conv5"

type: "Convolution"

bottom: "conv4"

top: "conv5"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 256

pad: 1

kernel\_size: 3

group: 2

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 1

}

}

}

layer {

name: "relu5"

type: "ReLU"

bottom: "conv5"

top: "conv5"

}

layer {

name: "pool5"

type: "Pooling"

bottom: "conv5"

top: "pool5"

pooling\_param {

pool: MAX

kernel\_size: 3

stride: 2

}

}

layer {

name: "fc6"

type: "InnerProduct"

bottom: "pool5"

top: "fc6"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

inner\_product\_param {

num\_output: 4096

weight\_filler {

type: "gaussian"

std: 0.005

}

bias\_filler {

type: "constant"

value: 1

}

}

}

layer {

name: "relu6"

type: "ReLU"

bottom: "fc6"

top: "fc6"

}

layer {

name: "drop6"

type: "Dropout"

bottom: "fc6"

top: "fc6"

dropout\_param {

dropout\_ratio: 0.5

}

}

layer {

name: "fc7"

type: "InnerProduct"

bottom: "fc6"

top: "fc7"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

inner\_product\_param {

num\_output: 4096

weight\_filler {

type: "gaussian"

std: 0.005

}

bias\_filler {

type: "constant"

value: 1

}

}

}

layer {

name: "relu7"

type: "ReLU"

bottom: "fc7"

top: "fc7"

}

layer {

name: "drop7"

type: "Dropout"

bottom: "fc7"

top: "fc7"

dropout\_param {

dropout\_ratio: 0.5

}

}

layer {

name: "fc8"

type: "InnerProduct"

bottom: "fc7"

top: "fc8"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

inner\_product\_param {

num\_output: 4

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

### detect.cpp

#include <opencv2/dnn.hpp>

#include <opencv2/imgproc.hpp>

#include <opencv2/highgui.hpp>

using namespace cv;

using namespace cv::dnn;

#include <fstream>

#include <iostream>

#include <cstdlib>

using namespace std;

//g++ -o detect detect.cpp -lopencv\_dnn -lopencv\_highgui -lopencv\_imgcodecs -lopencv\_imgproc -lstdc++ -lopencv\_core -L/usr/local/lib

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

{

String modelTxt="deploy.prototxt";

String modelBin="face\_locate\_iter\_2387.caffemodel";

Ptr<dnn::Importer> importer;

try

{

importer=dnn::createCaffeImporter(modelTxt,modelBin);

}

catch(const cv::Exception &err)

{

std::cerr<<err.msg<<std::endl;

}

if(!importer)

{

std::cerr<<"cant load network!"<<std::endl;

return 0;

}

dnn::Net net;

importer->populateNet(net);

importer.release();

Mat img = imread("000001.jpg",1);

Mat dst;

if(img.empty())

{

std::cerr<<"cant load image!"<<std::endl;

return 0;

}

resize(img,dst,Size(227,227));

dnn::Blob inputBlob = dnn::Blob(dst);

net.setBlob(".data",inputBlob);

net.forward();

dnn::Blob fc8=net.getBlob("fc8");

Mat fc8Mat = fc8.matRefConst().reshape(1,4);

std::cout << fc8Mat <<std::endl;

int x1 = fc8Mat.at<float>(0,0) \* 896;

int y1 = fc8Mat.at<float>(0,1) \* 592;

int x2 = fc8Mat.at<float>(0,2) \* 896;

int y2 = fc8Mat.at<float>(0,3) \* 592;

std::cout<<x1<<" "<<y1<<" "<<x2<<" "<<y2<<std::endl;

rectangle(img,Rect(Point(x1,y1),Point(x2,y2)),Scalar(255,0,0),2);

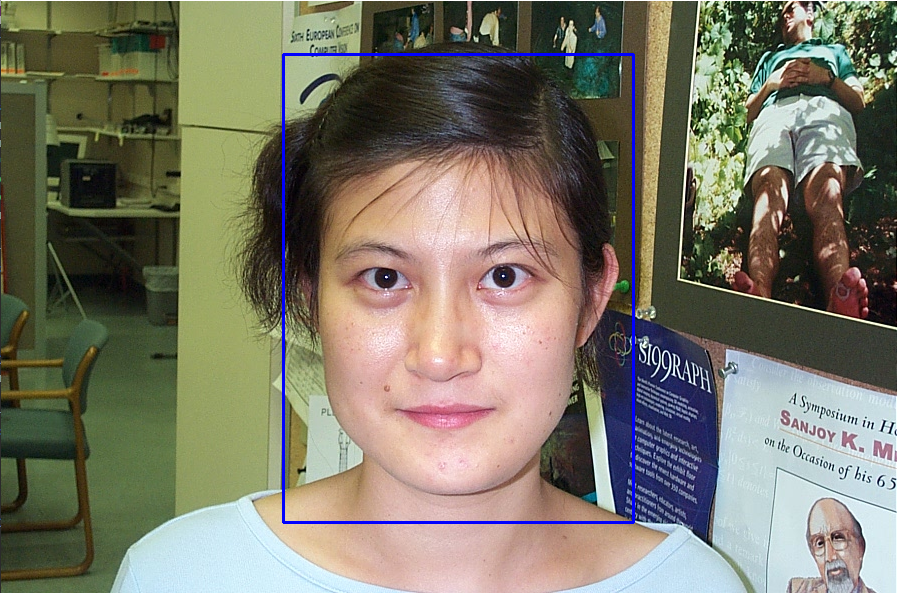
imshow("img",img);

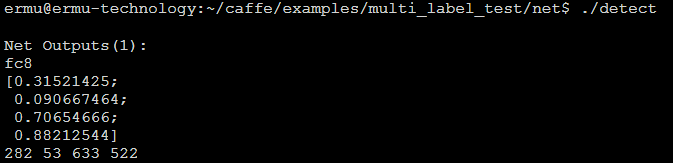
waitKey(0);

}

在这里注意获取某一层数据的方法。

### 测试结果





## 其它

在生成LMDB文件时，将图像归一化到227\*227，在训练过程中，遇到loss值为-nan的情况，将batch\_size改大成功解决。