# 搭建deepdetect环境

**deepdetect的github源码地址：** <https://github.com/beniz/deepdetect>

**mtcnn的github源码地址：**<https://github.com/DaFuCoding/MTCNN_Caffe>

## 步骤：

### 1.拉镜像命令：

docker pull beniz/deepdetect\_gpu

### 2.创建容器命令：

nvidia-docker run -d -p 4567:8080 -v /home:/root --name deepdetect\_gpu\_zj beniz/deepdetect\_gpu

### 3. 进入容器：

docker exec –it deepdetect\_gpu\_zj bash

**备注**：容器默认路径为/opt/deepdetect/build/main

### 4. 安装curl

**命令：**apt-get update

apt-get install curl

### 5. 启动deepdetect服务

cd /opt/deepdetect/build/main

./dede

### 6.获得服务信息：

**命令**：curl http://localhost:8080/info

**结果**：{"status":{"code":200,"msg":"OK"},"head":{"method":"/info","version":"0.1","branch":"maste

### 7. 创建服务：（图像分类）

**命令**：

curl -X PUT "http://localhost:8080/services/imageserv" -d

"{\"mllib\":\"caffe\",\"description\":\"image classification service\",\"type\":\"supervised\",\"parameters\":{\"input\":{\"connector\":\"image\"},\"mllib\":{\"nclasses\":1000}},\"model\":{\"repository\":\"/opt/models/ggnet/\"}}"

**结果：**

{"status":{"code":201,"msg":"Created"}}

### 8.图像分类

**命令**：

curl -X POST "http://localhost:8080/predict" -d "{\"service\":\"imageserv\",\"parameters\":{\"input\":{\"width\":224,\"height\":224},\"output\":{\"best\":3},\"mllib\":{\"gpu\":true}},\"data\":[\"http://i.ytimg.com/vi/0vxOhd4qlnA/maxresdefault.jpg\"]}"

**结果：**{"status":{"code":200,"msg":"OK"},"head":{"method":"/predict","time":852.0,"service":"imageserv"},"body":{"predictions":{"uri":"http://i.ytimg.com/vi/0vxOhd4qlnA/maxresdefault.jpg","classes":[{"prob":0.2255125343799591,"cat":"n03868863 oxygen mask"},{"prob":0.20917612314224244,"cat":"n03127747 crash helmet"},{"last":true,"prob":0.07399296760559082,"cat":"n03379051 football helmet"}]}}}

### 9.启动局域网内服务

由于deepdetect源码中的镜像，默认开启 –host 0.0.0.0 -port 8080，会导致我们二次开发时不能启动-host 0.0.0.0 –port 8080服务，所以重新创建镜像。(**备注**：可以更换端口号，比如：./dede –host 0.0.0.0 –port 4567 来启动服务)

**镜像名称**：deepdetect\_gpu\_host(默认不开启 –host 0.0.0.0 )

**生成容器**：nvidia-docker run -idt -p 9527:8080 -v /home:/home --name dede\_gpu\_lg deepdetect\_gpu\_host /bin/bash

(备注：9527端口映射到容器中的8080端口，容器默认开启8080端口，外部访问9527端口)

**使用方法（默认已编译成功）**：i、cd build/main

./dede –host 0.0.0.0

ii、详见三、put/post参数。

### 备注：

上面步骤可参考<https://hub.docker.com/r/beniz/deepdetect_gpu/>

镜像里面本身就带有源码，但是修改起来不太方便，所以可以在github上下载deepdetect的源码，再重新编译。编译步骤如下：

mkdir build

cd build

cmake ..

make

编译完成后可进入相应的目录，按照上面5-8步来操作。

# 二、修改deepdetect源码

以下是以mtcnn(人脸检测)的预测为例来修改deepdetect源码的备注。

## 获取图像

### 1）代码执行流程

1. caffelib.cc:: predict(const APIData &ad,APIData &out)

{

APIData cad = ad;

inputc.transform(cad);

}

2.caffeinputconns.h :: transform(const APIData &ad)

{

ImgInputFileConn::transform(ad); //在 imginputfileconn.h中

}

3.imginputfileconn.h:: transform (const APIData &ad)

{

DataEl<DDImg> dimg;

dimg.read\_element(u); //u为post时传入的参数“data”的值，也就是图片路径。

}

4. inputconnectorstrategy.h:: read\_element(const std::string &uri)

{

如果是本地图片：imginputfileconn.h:: read\_file(const std::string &fname);

如果是网络图片：imginputfileconn.h:: read\_mem(const std::string &content)；

该函数还支持base64(用字符串来代替图片数据流)

}

### 2）修改地方：

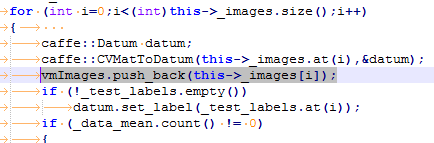
#### 第一处：获取图像信息

**第一步：定义**

在caffeinputconns.h中的CaffeInputInterface类，添加成员变量：std::vector<cv::Mat> vmImages;

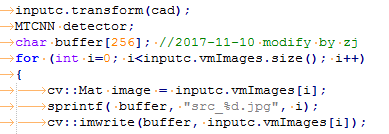
**第二步：赋值**

在caffeinputconns.h中的ImgCaffeInputFileConn类的transform(const APIData &ad)函数中给vmImages赋值。代码如下：



**第三步：使用**

在caffelib.cc中predict\_mtcnn(const APIData &ad,APIData &out)调用。代码如下：



#### 第二处：修改resize

**原因:** 源码中如果post时有输入宽高参数，就按照该宽高来缩放图像，如果没有就按照默

认参数224\*224来缩放。

**修改结果：**如果post时有输入宽高参数，就按照该宽高来缩放图像，如果没有就不进行缩

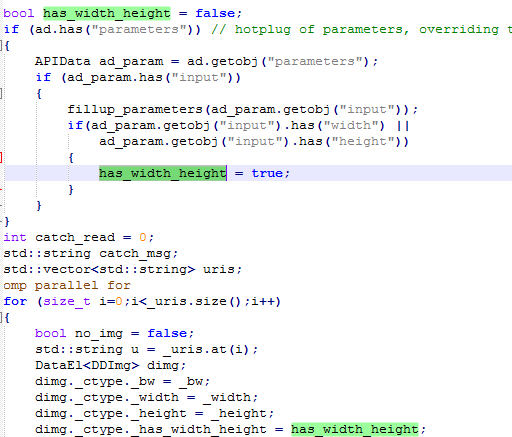
放。

**第一步：定义**

在imginputfileconn.h中的DDImg类中定义bool \_has\_width\_height=false;

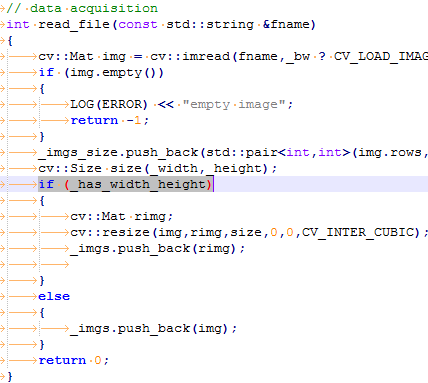
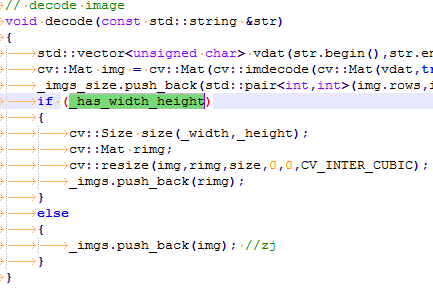
**第二步：赋值**

在imginputfileconn.h中的ImgInputFileConn类中的transform(const APIData &ad)函数中赋值，代码如下：



**第三步：使用**

在imginputfileconn.h中的DDImg类的函数read\_file(const std::string &fname)、decode(const std::string &str)添加判断，代码如下：（**备注**：读网络图片的read\_mem函数会调用decode）



## 构建网络

### 1）代码执行流程

发put请求时会构建网络。

1. **入口函数**：httpjsonapi.cc中

fillup\_response(response,\_hja->service\_create(sname,body),access\_log,code,tstart,accept\_encoding); 创建服务。

1. JDoc JsonAPI::service\_create(const std::string &sname, const std::string &jstr)

{

CaffeModel cmodel(ad\_model, nIndex); //创建对象并获得网络、模型文件路径

add\_service(…); //添加服务

}

1. service.h:: add\_service(…)

{

visitor\_init vi(ad); //会调用mllib.init(\_ad);

}

1. mlservice.h::init(const APIData &ad)

{

this->init\_mllib(ad.getobj("parameters").getobj("mllib"),nIndex); //

}

1. caffelib.cc:: init\_mllib(…)

{

create\_model\_mtcnn(true); //创建网络，该函数是修改了create\_model()函数。

}

**备注：**caffelib.cc:: create\_model\_mtcnn(…);

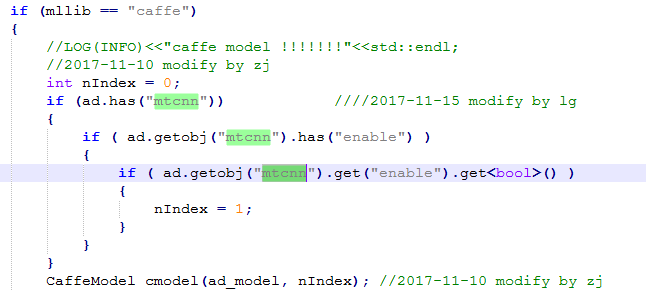
### 2）修改地方

**原因：**mtcnn人脸检测使用三个网络模型（p-net、r-net、o-net），deepdetect源码中只使用一个网络，所以需要修改相应的代码。

**修改结果：**可通过参数“mtcnn”来控制代码执行三个网络的mtcnn构建与预测。

#### 第一处：加“mtcnn”参数判断

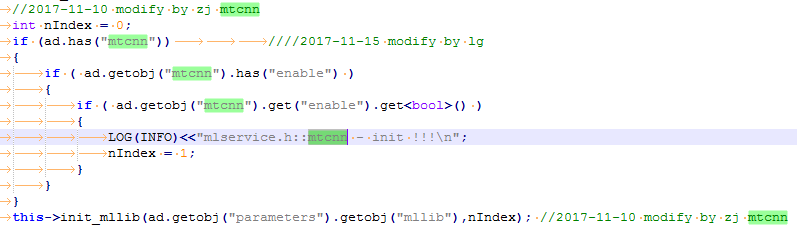
1. JDoc JsonAPI::service\_create(…)，修改代码如下：（nIndex==1为执行mtcnn）



**备注：**给caffemodel类的带参数构造函数CaffeModel(const APIData &ad, int nIndex=0);添加了一个int nIndex参数来控制是否执行mtcnn.

同时，在caffemodel类中添加一个read\_from\_repository\_mtcnn(…)函数，该函数的目的是为了获得网络、模型、超参数等的路径，并且该函数是仿着read\_from\_repository(…)改的。

1. mlservice.h::init(…)



**备注：**给caffelib类的init\_mllib(const APIData &ad, int nIndex)函数添加int nIndex参数来控制是否执行mtcnn. 当nIndex==1时，执行mtcnn.

同时，在caffelib类中添加函数create\_model\_mtcnn(const bool &test)，目的是创建mtcnn网络，该函数是仿着create\_model (const bool &test)改的。

## 预测

### 代码执行流程

发post请求时会执行预测。

1. **入口函数：**

httpjsonapi.cc中的

fillup\_response(response,\_hja->service\_predict(body),access\_log,code,tstart,accept\_encoding)

1. JsonAPI::service\_predict(const std::string &jstr)

{

this->predict(ad\_data,sname,out);

}

1. service.h:: predict(…)

{

visitor\_predict vp; //会调mllib.predict\_job(\_ad,\_out);

}

1. mlservice.h:: predict\_job(…)

{

this->predict\_mtcnn(ad,out); //

}

1. caffelib.cc:: predict\_mtcnn(…)

{

MTCNN detector;

detector.Detect(image,faceInfo,minSize,threshold, factor, \_net1p, \_net2r, \_net3o);

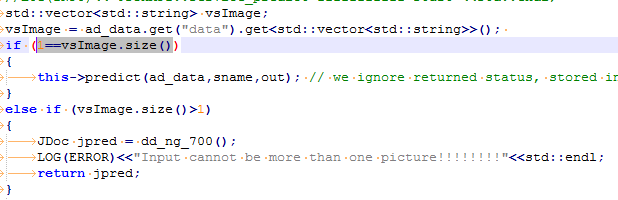
//执行mtcnn的检测

}

### 修改地方

#### 第一处：加多张图片判断

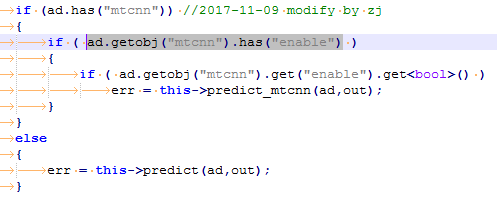
Jsonapi.cc 中的service\_predict(…)添加多张图片判断，目前代码只处理一张。修改代码如下：



**备注：**如果post时，“data”参数输入的图片个数大于1张，会返回700错误。700错误dd\_ng\_700()函数是自己添加的。

#### 第二处：添加mtcnn参数判断

mlservice.h:: predict\_job(…)中添加mtcnn参数判断，目的是判断是否执行mtcnn预测，**注意**：使用的是mtcnn参数中的enable参数来判断的，因为mtcnn预测还需要别的参数。修改代码如下：

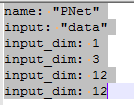
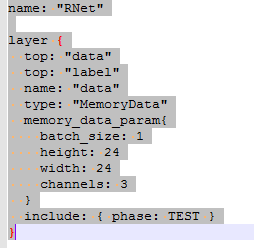


#### 第三处：修改mtcnn预测

该处需要修改的地方太多，就不一一说明，主要修改的地方如下：

##### 修改mtcnn的网络结构：

mtcnn的网络det1.prototxt、det2.prototxt、det3.prototxt需要修改的地方如下：（左改成右）

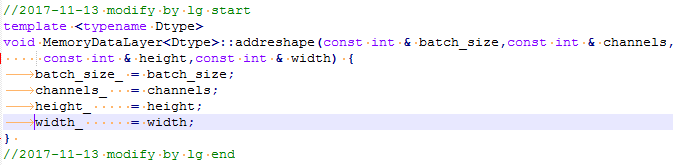
 

**原因：**左边使用Blob<float>\* crop\_input\_layer = net->input\_blobs()[0];来获取数据时会报错，需要改成右边，通过boost::shared\_ptr<MemoryDataLayer<float> > mem\_data\_layer;

mem\_data\_layer =

boost::static\_pointer\_cast<MemoryDataLayer<float> >(net->layers()[0]);来获取数据。

**但是**mtcnn的p-net输入的是图像金字塔（不同尺度的图像），所以需要将mem\_data\_layer进行reshape，不然会报错，所以需要在caffe源码的memory\_data\_layer类中添加addreshape(…), 代码如下：



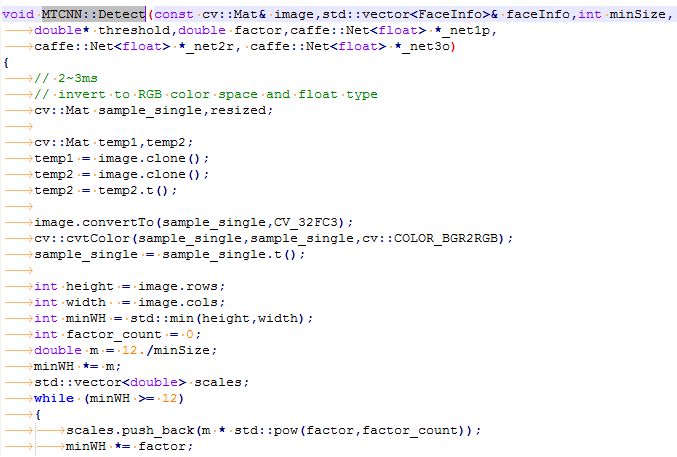
**备注：**在编译deepdetect时，默认是从

<https://github.com/beniz/caffe/archive/master.tar.gz>

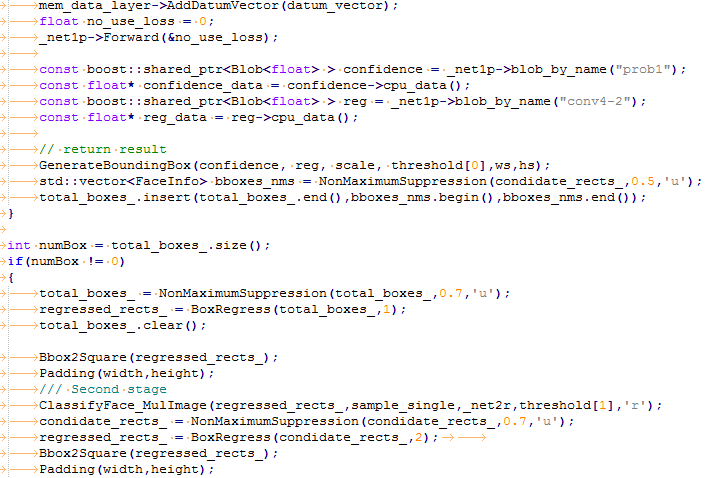
下载caffe后再编译的，所以修改源码后也会被下一次下载的给冲掉。最终的解决办法是：先fork beniz的caffe源码，再修改代码，同时将cmakelist.txt中下载caffe的地方都改成新github的地址，比如：

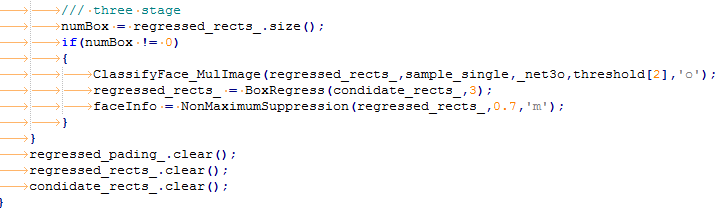
<https://github.com/lg12170226/caffe/archive/master.tar.gz>

##### mtcnn的检测代码：









**备注：**上面出现的函数有一部分修改过输入参数的类型。

# put\post命令参数

## put：

此指令只能在容器中运行：(启动deepdetect服务时：./dede )

curl -X PUT "http://localhost:8080/services/imageserv" -d

"{ \"mllib\":\"caffe\",

\"description\":\"image classification service\",

\"type\":\"supervised\",

\"parameters\":{ \"input\":{\"connector\":\"image\"},

\"mllib\":{\"nclasses\":2, \"gpu\":true, \”gpid\”:-1}

},

\"mtcnn\":{\”enable\”:true},

\"model\":{ \"repository\":\" /root/zhangjing/deepdetect/models/MTmodel/ \"}

}"

此指令能在局域网中运行：(启动deepdetect服务时：./dede -host 0.0.0.0)

curl -X PUT "http://192.168.15.100:9527/services/imageserv" -d

"{ \"mllib\":\"caffe\",

\"description\":\"image classification service\",

\"type\":\"supervised\",

\"parameters\":{ \"input\":{\"connector\":\"image\"},

\"mllib\":{\"nclasses\":2 , \"gpu\":true, \”gpid\”:-1}

},

\"mtcnn\":{\”enable\”:true},

\"model\":{ \"repository\":\" /root/zhangjing/deepdetect/models/MTmodel/ \"}

}"

**备注：**”mtcnn”中“enable”参数：true是为了执行mtcnn检测而加的。”gpid”为-1时，表示哪个gpu空闲使用哪个。如果没“gpu”参数，就使用cpu。

## post:

此指令只能在容器中运行：(启动deepdetect服务时：./dede )

curl -X POST "http://localhost:8080/predict" -d

"{ \"service\":\"imageserv\",

\"parameters\":{\"input\":{}

\"output\":{\"bbox\":true, \"confidence\_threshold\":0.1},

\"mllib\":{\"gpu\":true, \”gpid\”:-1}

},

\"mtcnn\":{\"enable":true,

\"minsize\":40,

\"factor\":0.709,

\"threshold\":[0.6,0.7,0.7]

}

\"data\":[\"http://i.ytimg.com/vi/0vxOhd4qlnA/maxresdefault.jpg\"]

}"

此指令能在局域网中运行：(启动deepdetect服务时：./dede -host 0.0.0.0)

curl -X POST " http://192.168.15.100:9527/predict" -d

"{ \"service\":\"imageserv\",

\"parameters\":{\"input\":{}

\"output\":{\"bbox\":true, \"confidence\_threshold\":0.1},

\"mllib\":{ \"gpu\":true, \”gpid\”:-1}

},

\"mtcnn\":{\"enable":true,

\"minsize\":40,

\"factor\":0.709,

\"threshold\":[0.6,0.7,0.7]

}

\"data\":[\"http://i.ytimg.com/vi/0vxOhd4qlnA/maxresdefault.jpg\"]

}"

**备注**：post：input参数中”width”、“height”参数都存在时，图像大小会根据”width”、“height”值来缩放；如果都不存在，图像大小不变；如果只存在一个参数，图像大小会缩放成(width,224)或(224,height)。对于大多数网络，输入图像大小必须与网络大小一致，不然会报错，但是对于mtcnn，最好使用原图大小。

# 错误

1. Put时，返回{"status":{"code":403,"msg":"Forbidden"}

表示服务已经启动，可以直接使用。

1. post时，返回{"status":{"code":400,"msg":"BadRequest","dd\_code":1006,"dd\_msg":"Service Bad Request Error"}}

表示输入参数有问题。

1. post时，返回{"status":{"code":700,"msg":"more pictures"}}

表示输入两张图片，目前代码只允许处理一张图片。

1. post时，{"status":{"code":400,"msg":"NotFound","dd\_code":1002,"dd\_msg":"Service Not Found"}} 表示没有预测服务
2. 执行./dede时，

error:

INFO - 00:57:51 - Running DeepDetect HTTP server on localhost:8080

ERROR - 00:57:51 - Error binding to socket.

terminate called without an active exception

Aborted (core dumped)

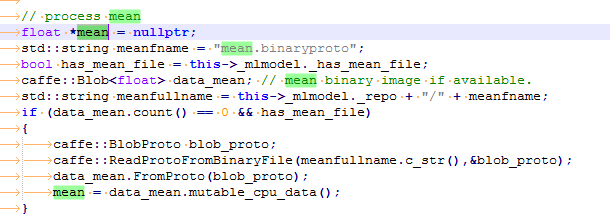
原因：dede服务已经启动

解决办法：重启容器

# mtcnn+emotion

## 获取均值

均值文件数据的获取代码在caffeinputconns.h::transform(const APIData &ad)中，代码如下：



均值数据保存在float \*mean中，该代码需要被放到预测函数中，同时均值处理需要放到原图resize、转float之后进行处理。Deepdetect的resize采用cv::INTER\_CUBIC方法，

