[Caffe1——Mnist数据集创建lmdb或leveldb类型的数据](http://blog.csdn.net/whiteinblue/article/details/45330801)

2015-04-28 09:00 623人阅读 [评论](http://blog.csdn.net/whiteinblue/article/details/45330801#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/whiteinblue/article/details/45330801#report)

[caffe源码解析](http://www.csdn.net/tag/caffe%e6%ba%90%e7%a0%81%e8%a7%a3%e6%9e%90)[lmdb数据转换](http://www.csdn.net/tag/lmdb%e6%95%b0%e6%8d%ae%e8%bd%ac%e6%8d%a2)[leveldb数据转换](http://www.csdn.net/tag/leveldb%e6%95%b0%e6%8d%ae%e8%bd%ac%e6%8d%a2)

目录[(?)[+]](http://blog.csdn.net/whiteinblue/article/details/45330801)

**Leveldb和lmdb简单介绍**

Caffe生成的数据分为2种格式：Lmdb和Leveldb。  
它们都是键/值对（Key/Value Pair）嵌入式数据库管理系统编程库。  
虽然lmdb的内存消耗是leveldb的1.1倍，但是lmdb的速度比leveldb快10%至15%，更重要的是lmdb允许多种训练模型同时读取同一组数据集。

因此lmdb取代了leveldb成为Caffe默认的数据集生成格式（http://blog.csdn.net/ycheng\_sjtu/article/details/40361947）

LevelDb有如下一些特点：  
　　　首先，LevelDb是一个持久化存储的KV系统，和Redis这种内存型的KV系统不同，LevelDb不会像Redis一样狂吃内存，而是将大部分数据存储到磁盘上。  
　　　其次，LevleDb在存储数据时，是根据记录的key值有序存储的，就是说相邻的key值在存储文件中是依次顺序存储的，而应用可以自定义key大小比较函数，LevleDb会按照用户定义的比较函数依序存储这些记录。  
　　　再次，像大多数KV系统一样，LevelDb的操作接口很简单，基本操作包括写记录，读记录以及删除记录。也支持针对多条操作的原子批量操作。

　　　另外，LevelDb支持数据快照（snapshot）功能，使得读取操作不受写操作影响，可以在读操作过程中始终看到一致的数据。

         除此外，LevelDb还支持数据压缩等操作，这对于减小存储空间以及增快IO效率都有直接的帮助。LevelDb性能非常突出，官方网站报道其随机写性能达到40万条记录每秒，而随机读性能达到6万条记录每秒。总体来说，LevelDb的写操作要大大快于读操作，而顺序读写操作则大大快于随机读写操作。至于为何是这样，看了我们后续推出的LevelDb日知录，估计您会了解其内在原因。（http://www.cnblogs.com/haippy/archive/2011/12/04/2276064.html）

**一：程序开始**

在Create.sh文件通过convert\_mnist\_data.bin来转换数据

**[plain]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. EXAMPLE=examples/mnist
2. DATA=data/mnist
3. BUILD=build/examples/mnist
4. ……
5. $BUILD/convert\_mnist\_data.bin $DATA/train-images-idx3-ubyte\
6. $DATA/train-labels-idx1-ubyte$EXAMPLE/mnist\_train\_${BACKEND} --backend=${BACKEND}

通过命令行解析（gflags）解析后，以上可以理解为在编译平台上（gcc等）运行convert\_mnist\_data.bin程序，程序需要4个参数：

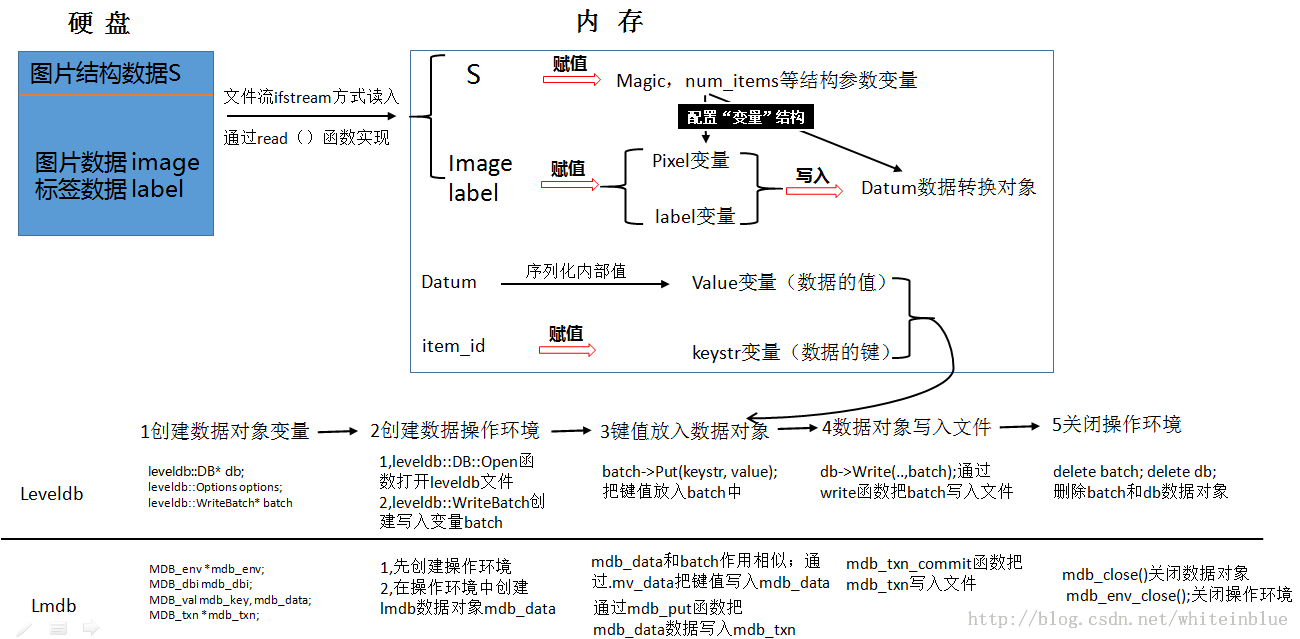
3个mian函数参数：1训练数据位置，2标签数据位置，3 lmdb数据存储位置。

1个程序中通过gflags宏定义的参数：转换的数据类型lmdb or leveldb。

convert\_mnist\_data.bin是由convert\_mnist\_data.cpp编译的可执行文件。

**二：数据转换流程图**

存放在硬盘中的mnist数据分为4个文件，训练和测试数据集，训练和测试标签集；其中数据集中存放了两类数据：图片结构数据和图片数据



**三：convert\_mnist\_data.cpp函数分析**

**1.引入必要的头文件和命名空间**

#include <gflags/gflags.h>//gflags命令行参数解析的头文件  
#include <glog/logging.h>//记录程序日志的glog头文件  
#include <google/protobuf/text\_format.h>//解析proto类型文件中，解析prototxt类型的头文件  
#include <leveldb/db.h>//引入leveldb类型数据头文件  
#include <leveldb/write\_batch.h>//引入leveldb类型数据写入头文件  
#include <lmdb.h>  
#include <stdint.h>  
#include <sys/stat.h>  
  
#include <fstream>  // NOLINT(readability/streams)  
#include <string>  
#include "caffe/proto/caffe.pb.h"//解析caffe中proto类型文件的头文件  
using namespace caffe;  // NOLINT(build/namespaces)  
using std::string;

**2.定义程序变**量backend

    通过宏定义字符串类型变量DEFINE\_stringbackend（这个是通过gflags来定义的变量，在程序调用时，通过--backend=${BACKEND}来给变量命名）

**3.main（）函数**

Argc为统计main函数接受的参数个数，正常调用时argc=4，argv为对应的参数值，

argv[1]=源数据路径，arg[2]=标签数据路径，arg[3]=保存lmdb数据的路径

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. **int** main(**int** argc, **char**\*\* argv)
2. {
3. **const** string& db\_backend = FLAGS\_backend; //获取--backend=${BACKEND}参数
4. **if** (argc != 4) {
5. gflags::ShowUsageWithFlagsRestrict(argv[0],
6. "examples/mnist/convert\_mnist\_data");
7. } **else** {
8. google::InitGoogleLogging(argv[0]);
9. convert\_dataset(argv[1], argv[2], argv[3], db\_backend);//函数功能把源数据装换成backend型数据，并保存在制定的路劲中
10. }
11. **return** 0;
12. }

**4. convert\_dataset（）函数**

**4.1读取源数据**

4.1.1打开源数据文件（文件先打开，才能读）

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. std::ifstream image\_file(image\_filename, std::ios::in | std::ios::binary);
2. std::ifstream label\_file(label\_filename, std::ios::in | std::ios::binary);
3. CHECK(image\_file) <<"Unable to open file "<< image\_filename;
4. CHECK(label\_file) <<"Unable to open file "<< label\_filename;

//引入std命名空间中的文件读入ifstream子空间，并创建“对象” image\_file（要读入的文件名，文件读入的方式），此处以二进制的方式读入image\_filename中的文件

//CHECK用于检测文件是否能够正常打开的函数，估计是定义在上面某个头文件里面的，具体哪个没有找到；感觉功能类似判断文件是否打开的函数image\_file.is\_open(）

4.1.2定义数据结构文件

根据mnist的图像结构，长，宽，channel，样本个数等

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. uint32\_t magic; //这个magic做什么的我也不清楚，程序读出来，CHECK后就没在使用
2. uint32\_t num\_items;
3. uint32\_t num\_labels;
4. uint32\_t rows;
5. uint32\_t cols;

//uint32\_t用typedef来自定义的一种数据类型，unsigned int32 ，每个int32整数占用4个字节

4.1.3读取图片结构数据

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. image\_file.read(**reinterpret\_cast**<**char**\*>(&magic), 4);
2. magic = swap\_endian(magic);//大端小端转换

//获取数据的结构信息，即图片的个数，width，height；这个数据的结果信息应该是一整型数据的方式存放在源数据的前n\*4个字节里面；label的n=2（magic和num\_labels），image的n=4（magic，num\_items，width，height）

//文件读取通过read函数来完成，read（读取内容的指针，读取的字节数），这里magic是一个int32类型的整数，每个占4个字节，所以这里指定为4

//reinterpret\_cast为C++中定义的强制转换符，这里把“&magic”，即magic的地址（一个16进制的数），转变成char类型的指针

**4.2创建lmdb和leveldb相关变量**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. //lmdb这个不太明白，只在 http://symas.com/mdb/doc/annotated.html上找了一些简单的介绍，见下问lmdb处
2. MDB\_env \*mdb\_env;
3. // Opaque structure for a database environment ；
4. MDB\_dbi mdb\_dbi;
5. MDB\_val mdb\_key, mdb\_data;
6. MDB\_txn \*mdb\_txn;
7. // leveldb
8. leveldb::DB\* db;//创建leveldb类型的指针
9. leveldb::Options options;
10. //感觉这个options应该是打开leveldb文件的方式，类似这种“存在就打开，不存在就创建”的文件打开方式
11. options.error\_if\_exists = **true**;// 存在就报错
12. options.create\_if\_missing = **true**;// 不存在就创建
13. options.write\_buffer\_size = 268435456; //256M
14. leveldb::WriteBatch\* batch = NULL;//创建leveldb类型的“实体数据”

**4.3 写入硬盘**

**Leveldb类型**

4.3.1打开（创建）数据库文件

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. LOG(INFO) << "Opening leveldb " << db\_path;
2. leveldb::Status status = leveldb::DB::Open(options, db\_path, &db);
3. CHECK(status.ok()) << "Failed to open leveldb " << db\_path<< ". Is it already existing?";
4. batch = **new** leveldb::WriteBatch();

//通过leveldb::DB::Open（）函数以options的方式，在db\_path路径下创建或者打开lmdb类型文件

4.3.2创建数据“转移”的中间变量

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. // Storing to db
2. **char** label;
3. **char**\* pixels = **new** **char**[rows \* cols];//定义char指针，指向字符串数组，字符串数组的容量为一个图片的大小
4. **int** count = 0;
5. **const** **int** kMaxKeyLength = 10; //最大的键值长度
6. **char** key\_cstr[kMaxKeyLength];

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. <span style="font-family: 'Microsoft YaHei';">string value; //用来获取“键”的内容</span>

//定义C类型的字符串，实际上是字符数组，因为二进制数据必须存储在固定长度的内存块里面，而C++中的string类型时没有固定的内存尺寸的；而变成字符数组后就有固定长度了

4.3.3创建“转换”数据对象datum

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. //设置datum数据对象的结构，其结构和源图像结构相同
2. Datum datum;
3. datum.set\_channels(1);
4. datum.set\_height(rows);
5. datum.set\_width(cols);

4.3.4读取源数据值并“赋值”给datum

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. image\_file.read(pixels, rows \* cols); //从数据中读取rows \* cols个字节，图像中一个像素值（应该是int8类型）用一个字节表示即可
2. label\_file.read(&label, 1);//读取标签
3. datum.set\_data(pixels, rows\*cols);//setdata函数把源图像值放入，datum对象
4. datum.set\_label(label);//set\_label函数把标签值放入datum
5. //snprintf(str1,size\_t,"format",str),把str按照format的格式以字符串的形式写入str1，size\_t,表示写入的字符个数
6. //这里是把item\_id转换成8位长度的十进制整数，然后在变成字符串复制给key\_str，如：item\_id=1500（int），则key\_cstr=00015000（string,\0为字符串结束标志）
7. snprintf(key\_cstr, kMaxKeyLength, "%08d", item\_id);
8. datum.SerializeToString(&value);
9. //感觉是将datum中的值序列化成字符串，保存在变量value内，通过指针来给value赋值
10. string keystr(key\_cstr);

4.3.5将数据写入db数据对象batch中

    batch->Put(keystr, value);//通过batch中的子方法Put，把数据写入datum中（此时在内存中）

4.3.6把db数据写入硬盘

    代码选择1000个样本放入一个batch中，通过batch以批量的方式把数据写入硬盘；写入硬盘通过db.write（）函数来实现。

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. **if** (++count % 1000 == 0) {//每个batch为1000个样本
2. // Commit txn
3. **if** (db\_backend == "leveldb") {  // leveldb
4. db->Write(leveldb::WriteOptions(), batch);
5. **delete** batch;
6. batch = **new** leveldb::WriteBatch();

//把batch写入到db中，然后删除batch并重新创建，这里为什么要删除重建有些不理解；删除可能是为了清理变量，减少内存占用吧，之后又重建了。

4.3.7写入最后一个batch

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. **if** (count % 1000 != 0) {
2. **if** (db\_backend == "leveldb") {  // leveldb
3. db->Write(leveldb::WriteOptions(), batch);
4. **delete** batch;
5. **delete** db;//删除临时变量，清理内存占用

**Lmdb类型**

**变量和函数说明**

MDB\_dbi :在数据库环境中的一个独立的数据句柄

MDB\_env：数据库环境的“不透明结构”，不透明类型是一种灵活的类型，他的大小是未知的

MDB\_val：用于从数据库输入输出的通用结构

MDB\_txn：不透明结构的处理句柄，所有的数据库操作都需要处理句柄，处理句柄可指定为只读或读写

**mdb\_env\_create(**[**MDB\_env**](http://symas.com/mdb/doc/group__internal.html#structMDB__env)**\*\* *env*)：**

         创建一个lmdb环境句柄，此函数给mdb\_env结构分配内存；释放内存或者关闭句柄可以通过mdb\_env\_close（）函数来操作。在使用meb\_env\_create（）句柄前，必须使用ndb\_env\_open（）函数打开。

         参数：env 新句柄的存储地址

**mdb\_env\_open(MDB\_env \* env,const char \* path,unsigned int flags,mdb\_mode\_t mode )**

打开环境句柄，

          参数：1 env，是mdb\_env\_create（）函数返回的环境句柄

                    2 path，数据库文件隶属的文件夹，文件夹必须存在而且是可读的。

**mdb\_env\_set\_mapsize    (MDB\_env \*env ,  size\_t size )**

        设置当前环境的内存映射（内存地图）的尺寸。

**int mdb\_txn\_begin (MDB\_env \*  env,  MDB\_txn \*  parent, unsigned int flags,  MDB\_txn \*\*  txn )**

        在环境内创建一个用来使用的“处理”transaction句柄

        参数：1，env，环境

         4，MDB\_txn\*\* txn 新txn句柄存储的地址

**mdb\_open**

通过宏定义的方式，把mdb\_open（）函数用msb\_dbi\_open（）函数替代

#define  mdb\_open(txn, name, flags,dbi )      mdb\_dbi\_open(txn,name,flags,dbi)

**mdb\_dbi\_open(txn,name,flags,dbi)**

在环境中打开一个数据库

参数：

1，txn  mdn\_txn\_begin（）函数返回的处理句柄

2，name 要打开的数据库名称， 如果环境中只需要一个单独的数据库，这个值为null

3，flags 指定当前数据库的操作选项

4，dbi 新的mdb\_dbi句柄存储的地址

**int mdb\_put (MDB\_txn \*        txn,MDB\_dbi           dbi,MDB\_val\* key,MDB\_val \* data,unsigned int flags )**

把数据条目保存到数据库；函数把key/data（键值对）保存到数据库

参数：

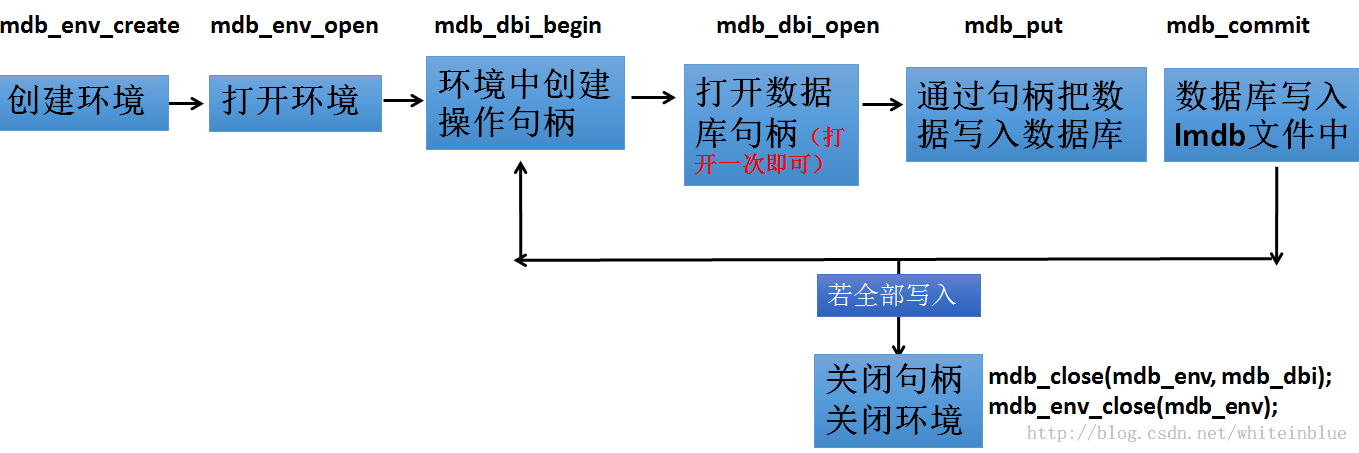
         1，txn   mdb\_txn\_begin（）函数返回的transaction处理句柄

         2，dbi  mdb\_dbi\_open() 函数返回的数据库句柄

         3，key   4，data

**int mdb\_txn\_commit (  MDB\_txn \*     txn   )**

提交所有transaction操作到数据库中；交易句柄必须是“自由的”freed；在本次调用之后，他和它本身的“光标（指针）”不能够被在此使用；需要再一次指定txn



5.3.1创建lmdb操作环境（输入输出环境）

1）创建lmdb操作环境，

2）设置环境参数，

3）在存储位置“打开”lmdb环境，

4）在环境内创建一个用来使用的“处理”transaction句柄

5）打开lmdb类型文件

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. LOG(INFO) <<"Opening lmdb "<< db\_path;
2. CHECK\_EQ(mkdir(db\_path, 0744), 0)
3. <<"mkdir "<< db\_path <<"failed";//感觉是，检查文件路径的
4. CHECK\_EQ(mdb\_env\_create(&mdb\_env), MDB\_SUCCESS) <<"mdb\_env\_create failed";//感觉是创建lmdb类型数据的操作环境，并检查
5. CHECK\_EQ(mdb\_env\_set\_mapsize(mdb\_env, 1099511627776), MDB\_SUCCESS)
6. // 1TB，感觉是设置lmdb类型操作环境参数
7. <<"mdb\_env\_set\_mapsize failed";
8. CHECK\_EQ(mdb\_env\_open(mdb\_env, db\_path, 0, 0664), MDB\_SUCCESS)
9. //感觉是在db\_path处打开上面创建的操作环境
10. <<"mdb\_env\_open failed";
11. CHECK\_EQ(mdb\_txn\_begin(mdb\_env, NULL, 0, &mdb\_txn), MDB\_SUCCESS)
12. //提交所有transaction操作到数据库中
13. <<"mdb\_txn\_begin failed";
14. CHECK\_EQ(mdb\_open(mdb\_txn, NULL, 0, &mdb\_dbi), MDB\_SUCCESS)
15. //<span style="font-family: Arial, Helvetica, sans-serif;">在环境中打开一个数据库</span>
16. <<"mdb\_open failed. Does the lmdb already exist? ";

5.3.2创建数据“转移”的中间变量

5.3.3创建“转换”数据对象datum

5.3.4读取源数据值并“赋值”给datum

    见4.3.2,4.3.3,4.3.4

5.3.5把数据放入lmdb数据类型对象mdb\_data（MDB\_val类型）

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. {  // lmdb
2. //mv感觉应该是move value，应该是和write（）和read（）函数文件读写的方式一样，以固定的字节长度按照地址进行读写操作
3. mdb\_data.mv\_size = value.size();//获取value的字节长度，类似sizeof（）函数
4. mdb\_data.mv\_data = **reinterpret\_cast**<**void**\*>(&value[0]);//把value的首个字符地址传换成空类型的指针
5. mdb\_key.mv\_size = keystr.size();
6. mdb\_key.mv\_data = **reinterpret\_cast**<**void**\*>(&keystr[0]);
7. //通过mdb\_put函数把mdb\_key和mdb\_data所指向的数据，写入到mdb\_dbi（mdb\_dbi个人理解，这个貌似有问题）

5.3.6 lmdb数据类型对象写入mdb\_txn中

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. CHECK\_EQ(mdb\_put(mdb\_txn, mdb\_dbi, &mdb\_key, &mdb\_data, 0), MDB\_SUCCESS)<<"mdb\_put failed";

5.3.7lmdb写入到硬盘

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. 感觉是通过mdb\_txn\_commit函数把mdb\_txn中的数据写入到硬盘
2. CHECK\_EQ(mdb\_txn\_commit(mdb\_txn), MDB\_SUCCESS)<<"mdb\_txn\_commit failed";

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. CHECK\_EQ(mdb\_txn\_begin(mdb\_env, NULL, 0, &mdb\_txn), MDB\_SUCCESS)<<"mdb\_txn\_begin failed";

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. //重新设置mdb\_txn的写入位置，类似文件写入时的app方式，就是追加（继续）写入

5.3.8写入最后一个batch

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. CHECK\_EQ(mdb\_txn\_commit(mdb\_txn), MDB\_SUCCESS) <<"mdb\_txn\_commit failed";
2. mdb\_close(mdb\_env, mdb\_dbi);//关闭mdb数据对象变量
3. mdb\_env\_close(mdb\_env);//关闭mdb操作环境变量

**四：大端小端转换**

CPU处理器对多字节数据的存储方式，对二进制文件的可移植性有着决定性的影响；二进制文件里数据的排列顺序与他们在计算机内存的存储顺序完全一样。大端字节的计算机，数据的最高位存储在最前面；小端字节的计算机上数据的最低位存储在最前面；大端字节计算机上存储的二进制文件无法在小端计算机上正确读取，反之亦然。感觉mnist的数据集在制作存储的时候官方采用的CPU的存储方式可能和我们的CPU不一样，所以低于mnist需要进行大端小端的转换。

详细介绍参考：http://www.cnblogs.com/passingcloudss/archive/2011/05/03/2035273.html

//convert big endian to little endian in C ;http://stackoverflow.com/questions/2182002/convert-big-endian-to-little-endian-in-c-without-using-provided-funcuint32\_t

//大端小端转换（大端小端为一种字节顺序存储的方式，不同的CPU有不同的存储方式）

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45330801)[在CODE上查看代码片](https://code.csdn.net/snippets/654552)

1. uint32\_t swap\_endian(uint32\_t val)
2. {//<<为位操作符，“<<”左移一位，实际数值乘以2，整形数字4，对应二进制为：……010，4<<2 ……01000，左移两位后，变成16
3. val = ((val << 8) & 0xFF00FF00) | ((val >> 8) & 0xFF00FF); //变量之间的“&”为按照“位”，进行与操作，二进制数：1010 & 0110 =0010
4. **return** (val << 16) | (val >> 16);// 变量之间的“|”操作符为按照“位”进行或操作，二进制数：1010 & 0110 =1110
5. }

五：以上代码注释为个人理解，如有遗漏，错误还望大家多多交流，指正，以便共同学习，进步！！

[Caffe2——cifar10数据集创建lmdb或leveldb类型的数据](http://blog.csdn.net/whiteinblue/article/details/45367097)

2015-04-29 15:20 532人阅读 [评论](http://blog.csdn.net/whiteinblue/article/details/45367097#comments)(1) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/whiteinblue/article/details/45367097#report)

[caffe](http://www.csdn.net/tag/caffe)[lmdb](http://www.csdn.net/tag/lmdb)

目录[(?)[+]](http://blog.csdn.net/whiteinblue/article/details/45367097)

cifar10数据集和mnist数据集存储方式不同，cifar10数据集把标签和图像数据以bin文件的方式存放在同一个文件内，这种存放方式使得每个子cifar数据bin文件的结构相同，所以cifar转换数据代码比mnist的代码更加的模块化，分为源数据读取模块（image\_read函数），把lmdb（leveldb）数据转换的变量声明，句柄（函数）调用都放到定义的caffe：：db子空间中，这样简化了代码，而且使得代码更加清晰。

**一：程序开始**

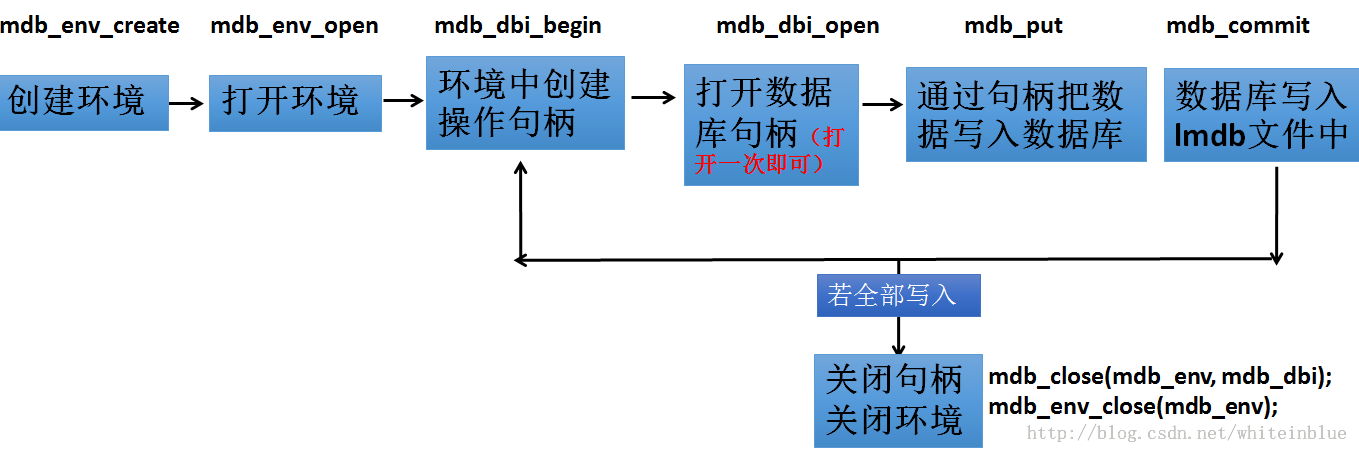
和转换mnist数据不同的是，cifar并没有使用gflags命令行解析工具；所以也没有通过gflags的宏定义来指定要转换的数据类型，而是把转换的类型参数直接作为main（）函数的参数（这种方式便于理解）。

在Create.sh文件中，调用convert\_cifar\_data.bin语句为：

./build/examples/cifar10/convert\_cifar\_data.bin$DATA $EXAMPLE $DBTYPE

convert\_cifar\_data.bin程序，程序需要3个参数，分别为源数据路径，lmdb（leveldb）存储路径，要转换的数据类型lmdb or leveldb

**二：数据转换流程图**



**三：convert\_cifar\_data.cpp函数分析**

**1引入必要的头文件和命名空间**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45367097)[在CODE上查看代码片](https://code.csdn.net/snippets/655717)

1. #include <fstream>
2. #include <string>
3. #include "boost/scoped\_ptr.hpp"
4. #include "glog/logging.h"
5. #include "google/protobuf/text\_format.h"
6. #include "stdint.h"
7. #include "caffe/proto/caffe.pb.h"
8. #include "caffe/util/db.hpp"

头文件和convert\_mnist\_data.cpp的区别：

1，没有引入gflags命令行解析工具；

2，没有引入leveldb和lmdb的数据头文件

3，引入了"boost/scoped\_ptr.hpp"智能指针头文件

4，引入"caffe/util/db.hpp"头文件，里面包装了对lmdb和leveldb数据对象的操作内容

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45367097)[在CODE上查看代码片](https://code.csdn.net/snippets/655717)

1. **using** caffe::Datum;
2. **using** boost::scoped\_ptr;
3. **using** std::string;
4. **namespace** db = caffe::db;

命名空间区别：

1，没有引入全部caffe命名空间，而是局部引入了两个caffe命名空间下的子空间 caffe::Datum和caffe::db

2，引入boost::scoped\_ptr;智能指针命名空间，智能指针，它能够保证在离开作用域后对象被自动释放；在mnist数据转换代码中，经常出现delete batch等删除临时变量的指令，通过智能指针可以自动删除过期的变量，对于控制程序内存占用很实用。

**2 main（）函数**

接收参数，调用转换函数convet\_dataset（）

**3 convet\_dataset（）函数**

**3.1以智能指针的方式创建db::DB类型的对象 train\_db**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45367097)[在CODE上查看代码片](https://code.csdn.net/snippets/655717)

1. scoped\_ptr<db::DB>  train\_db(db::GetDB(db\_type));

智能指针的创建方式类似泛型的格式，上面通过db.cpp内定义的命名的子命名空间中db的“成员函数”GetDB函数来初始化train\_db对象

**3.2 创建lmdb数据对象**

**3.2.1创建环境；设置环境参数，打开环境**

调用tran\_db对象的open方法，以“对db::NEW的方式，创建lmdb（leveldb）类型文件

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45367097)[在CODE上查看代码片](https://code.csdn.net/snippets/655717)

1. train\_db->Open(output\_folder+ "/cifar10\_train\_" + db\_type,db::NEW);

db命名空间中open函数具体实现代码：

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45367097)[在CODE上查看代码片](https://code.csdn.net/snippets/655717)

1. **void** LMDB::Open(**const** string& source, Mode mode) {
2. MDB\_CHECK(mdb\_env\_create(&mdb\_env\_));//创建lmdb操作环境
3. MDB\_CHECK(mdb\_env\_set\_mapsize(mdb\_env\_, LMDB\_MAP\_SIZE));//设置环境内训映射
4. **if** (mode == NEW) {
5. CHECK\_EQ(mkdir(source.c\_str(),0744), 0) << "mkdir " << source <<"failed";
6. }//检查文件
7. **int** flags = 0;
8. **if** (mode == READ) {
9. flags = MDB\_RDONLY | MDB\_NOTLS;
10. }
11. MDB\_CHECK(mdb\_env\_open(mdb\_env\_,source.c\_str(), flags, 0664));//打开创建的环境
12. LOG(INFO) << "Openedlmdb " << source;
13. }

**3.2.2创建并打开transaction操作句柄，打开数据库句柄**

调用db命名空间中的Transaction方法，来创建句柄对象txn

scoped\_ptr<db::Transaction>  txn(train\_db->NewTransaction());

db命名空间中NewTransaction（）函数代码

//在lmdb环境中创建操作句柄

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45367097)[在CODE上查看代码片](https://code.csdn.net/snippets/655717)

1. LMDBTransaction\* LMDB::NewTransaction() {
2. MDB\_txn\* mdb\_txn;
3. MDB\_CHECK(mdb\_txn\_begin(mdb\_env\_,NULL, 0, &mdb\_txn));//创建操作句柄
4. MDB\_CHECK(mdb\_dbi\_open(mdb\_txn,NULL, 0, &mdb\_dbi\_));//打开数据库环境
5. **return** **new** LMDBTransaction(&mdb\_dbi\_,mdb\_txn);
6. }

**3.3 定义数据结构文件**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45367097)[在CODE上查看代码片](https://code.csdn.net/snippets/655717)

1. **const** **int** kCIFARSize =32;
2. **const** intkCIFARImageNBytes = 3072; //32\*32=1024，RGB各占一个字节,感觉应该为uint8\_t，0~255，
3. **const** intkCIFARBatchSize = 10000; //cifar共计5万个训练样本，分成5份batches，每份1万个
4. **const** **int** kCIFARTrainBatches= 5;
5. // Data buffer
6. **int** label;
7. charstr\_buffer[kCIFARImageNBytes]; //定义字符数组，一个数组可以存放一张图片的数据
8. Datum datum;
9. datum.set\_channels(3);
10. datum.set\_height(kCIFARSize);
11. datum.set\_width(kCIFARSize);

**3.4 打开源数据文件**

下载的Cifar数据存放在6个bin文件内，从data\_batch\_1.bin到data\_batch\_5.bin；本文以循环的方式分别读取每个bin文件。每个bin文件存储1万张图片

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45367097)[在CODE上查看代码片](https://code.csdn.net/snippets/655717)

1. **for** (**int** fileid = 0;fileid < kCIFARTrainBatches; ++fileid) {
2. snprintf(str\_buffer, kCIFARImageNBytes, "/data\_batch\_%d.bin", fileid + 1);
3. std::ifstream data\_file((input\_folder + str\_buffer).c\_str(),std::ios::in| std::ios::binary);
4. CHECK(data\_file) << "Unable to open train file #" <<fileid + 1;
5. //str\_buffer=/data\_batch\_1.bin,等等，但str\_buffer是个字符数组
6. //以二进制和流输入的方式打开文件data/cifar10/data\_batch\_1.bin
7. //c\_str() 以 char\* 形式传回 string 内含字符串

**3.5 读取源数据文件**

和mnist不同的是，mnist源数据集有4个文件；mnist读取数据时，分别调用文件读取函数read（），感觉这是由于mnist源数据中label数据和image数据中存储的内容不统一，image文件中除了存储图像数据外，还存储了图像结构数据；而图像结构数据和图像数据读取的方式不一样，而且还涉及到大端小端的转换；所以没有定义一个统一的图像读取函数来读取；本项由于image和标签数据都存储在同一个bin文件中，所以可以定义统一的图片读取函数read\_image来读取源数据内容。

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45367097)[在CODE上查看代码片](https://code.csdn.net/snippets/655717)

1. **for** (**int** itemid = 0;itemid < kCIFARBatchSize; ++itemid) {
2. read\_image(&data\_file, &label,str\_buffer);
3. //调用read\_image函数从.bin文件读取数据，通过指针赋值给label和str\_buffer
4. **void** read\_image(std::ifstream\* file,**int**\* label, **char**\*buffer) {
5. charlabel\_char;
6. file->read(&label\_char, 1);
7. //读取label\_char的内容；CIFAR10数据应该是一个类似结构体的数据对，有label和data两个属性，其中label用label\_char来定义的
8. \*label = label\_char; //把label\_char的值，给label
9. file->read(buffer,kCIFARImageNBytes);
10. **return**;
11. }

**3.6 读取的数据赋值到“转换”数据对象datum，并序列化**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45367097)[在CODE上查看代码片](https://code.csdn.net/snippets/655717)

1. datum.set\_label(label);
2. datum.set\_data(str\_buffer,kCIFARImageNBytes);
3. string out;
4. CHECK(datum.SerializeToString(&out));

**3.7 把数据写入数据库**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45367097)[在CODE上查看代码片](https://code.csdn.net/snippets/655717)

1. **int** length =snprintf(str\_buffer, kCIFARImageNBytes, "%05d",fileid \*kCIFARBatchSize + itemid);

//上一行代码有两个作用：

1，把fileid \* kCIFARBatchSize + itemid的值赋值给str\_buffer，此处的赋值为每个样本（图片）的id，

2，给length赋值，此处length=5

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45367097)[在CODE上查看代码片](https://code.csdn.net/snippets/655717)

1. string out;
2. txn->Put(string(str\_buffer, length),out);//string(str\_buffer, length)用来截取str\_buffer的前length个字符；

//db命名空间中，Put函数代码；

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45367097)[在CODE上查看代码片](https://code.csdn.net/snippets/655717)

1. **void** LMDBTransaction::Put(conststring& key,**const** string& value) {
2. MDB\_val mdb\_key, mdb\_value;//声明MDB\_val不透明类型数据结构“对象”
3. mdb\_key.mv\_data = **const\_cast**<**char**\*>(key.data());//通过指针的方式给mdb\_key赋值
4. mdb\_key.mv\_size = key.size();
5. mdb\_value.mv\_data = **const\_cast**<**char**\*>(value.data());
6. mdb\_value.mv\_size = value.size();
7. MDB\_CHECK(mdb\_put(mdb\_txn\_, \*mdb\_dbi\_,&mdb\_key, &mdb\_value, 0));
8. //通过mdb\_put（）句柄把mdb\_key和mdb\_value中的数据，写入数据库中
9. }

**3.8 把数据库写入lmdb文件并关闭写入环境**

//这个commit函数和close函数，不是在caffe：db命名空间中定义的函数，估计是caffe命名空间中自带的函数。

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45367097)[在CODE上查看代码片](https://code.csdn.net/snippets/655717)

1. txn->Commit();
2. train\_db->Close();

**3.9用上面类似的方法把测试集写入lmdb文件中**

**四，相关文件**

convert\_cifar10\_data.cpp文件

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45367097)[在CODE上查看代码片](https://code.csdn.net/snippets/655717)

1. // This script converts the CIFAR dataset to the leveldb format used
2. // by caffe to perform classification.
3. // Usage:
4. //    convert\_cifar\_data input\_folder output\_db\_file
5. // The CIFAR dataset could be downloaded at
6. //    http://www.cs.toronto.edu/~kriz/cifar.html
8. #include <fstream>  // NOLINT(readability/streams)，文件输入输出必备的文件流
9. #include <string>
11. #include "boost/scoped\_ptr.hpp"//智能指针
12. #include "glog/logging.h"//用于日志记录，具体记录什么不是很清楚，
13. #include "google/protobuf/text\_format.h"//用于解析.prototxt文件的
14. #include "stdint.h"
16. #include "caffe/proto/caffe.pb.h" //解析.prototxt文件的头文件
17. #include "caffe/util/db.hpp" //db.cpp文件中定义了NewTransaction（），Open（）等leveldb和lmdb操作函数
19. **using** caffe::Datum;
20. **using** boost::scoped\_ptr;//是一个简单的智能指针，它能够保证在离开作用域后对象被自动释放。
21. **using** std::string;
22. **namespace** db = caffe::db;//引入caffe命名空间中的db子命名空间
24. **const** **int** kCIFARSize = 32;
25. **const** **int** kCIFARImageNBytes = 3072;//32\*32=1024，RGB各占一个字节,感觉应该为uint8\_t，0~255，
26. **const** **int** kCIFARBatchSize = 10000;//cifar共计5万个训练样本，分成5份batches，每份1万个，
27. **const** **int** kCIFARTrainBatches = 5;

30. **void** read\_image(std::ifstream\* file, **int**\* label, **char**\* buffer) {
31. **char** label\_char;
32. file->read(&label\_char, 1);//读取label\_char的内容；CIFAR10数据应该是一个类似结构体的数据对，有label和data两个属性，其中label用label\_char来定义的
33. \*label = label\_char;//把label\_char的值，给label
34. file->read(buffer, kCIFARImageNBytes);
35. **return**;
36. }
38. //以值引用的方式传递参数（string& input\_folder），
39. **void** convert\_dataset(**const** string& input\_folder, **const** string& output\_folder,
40. **const** string& db\_type) {
41. scoped\_ptr<db::DB> train\_db(db::GetDB(db\_type));//以智能指针的方式创建db::DB类型的对象 train\_db ，这个db::DB是什么东西有些不清楚，db.cpp中并没有发现这个DB类型的命名空间。
42. train\_db->Open(output\_folder + "/cifar10\_train\_" + db\_type, db::NEW);//调用tran\_db对象的open方法，以“对db::NEW的方式，创建（或打开）文件
43. scoped\_ptr<db::Transaction> txn(train\_db->NewTransaction());//这个transaction暂时不清楚是干什么用的
44. // Data buffer
45. **int** label;
46. **char** str\_buffer[kCIFARImageNBytes];//定义字符数组，一个数组可以存放一张图片的数据
47. Datum datum;
48. datum.set\_channels(3);
49. datum.set\_height(kCIFARSize);
50. datum.set\_width(kCIFARSize);
52. LOG(INFO) << "Writing Training data";
53. **for** (**int** fileid = 0; fileid < kCIFARTrainBatches; ++fileid) {//依次遍历每个batches，共计5个
54. // Open files
55. LOG(INFO) << "Training Batch " << fileid + 1;
56. snprintf(str\_buffer, kCIFARImageNBytes, "/data\_batch\_%d.bin", fileid + 1); //str\_buffer=/data\_batch\_1.bin,等等，但str\_buffer是个字符数组
57. std::ifstream data\_file((input\_folder + str\_buffer).c\_str(),//以二进制和流输入的方式打开文件data/cifar10/data\_batch\_1.bin
58. std::ios::in | std::ios::binary);//c\_str() 以 char\* 形式传回 string 内含字符串
59. CHECK(data\_file) << "Unable to open train file #" << fileid + 1;
60. **for** (**int** itemid = 0; itemid < kCIFARBatchSize; ++itemid) {
61. read\_image(&data\_file, &label, str\_buffer);//调用read\_image函数从.bin文件读取数据，给label和str\_buffer赋值
62. datum.set\_label(label);
63. datum.set\_data(str\_buffer, kCIFARImageNBytes);
64. **int** length = snprintf(str\_buffer, kCIFARImageNBytes, "%05d",
65. fileid \* kCIFARBatchSize + itemid);//给str\_buffer赋值，此处的赋值为每个样本（图片）的id，length=5；其实是把str\_buffer的前5个字符赋值为id
66. string out;
67. CHECK(datum.SerializeToString(&out));
68. txn->Put(string(str\_buffer, length), out);//string(str\_buffer, length)用来截取str\_buffer的前length个字符；
69. }
70. }
71. txn->Commit();
72. train\_db->Close();
74. LOG(INFO) << "Writing Testing data";
75. scoped\_ptr<db::DB> test\_db(db::GetDB(db\_type));
76. test\_db->Open(output\_folder + "/cifar10\_test\_" + db\_type, db::NEW);
77. txn.reset(test\_db->NewTransaction());
78. // Open files
79. std::ifstream data\_file((input\_folder + "/test\_batch.bin").c\_str(),
80. std::ios::in | std::ios::binary);
81. CHECK(data\_file) << "Unable to open test file.";
82. **for** (**int** itemid = 0; itemid < kCIFARBatchSize; ++itemid) {
83. read\_image(&data\_file, &label, str\_buffer);
84. datum.set\_label(label);
85. datum.set\_data(str\_buffer, kCIFARImageNBytes);
86. **int** length = snprintf(str\_buffer, kCIFARImageNBytes, "%05d", itemid);
87. string out;
88. CHECK(datum.SerializeToString(&out));
89. txn->Put(string(str\_buffer, length), out);
90. }
91. txn->Commit();
92. test\_db->Close();
93. }
95. **int** main(**int** argc, **char**\*\* argv) {
96. **if** (argc != 4) {
97. printf("This script converts the CIFAR dataset to the leveldb format used\n"
98. "by caffe to perform classification.\n"
99. "Usage:\n"
100. "    convert\_cifar\_data input\_folder output\_folder db\_type\n"
101. "Where the input folder should contain the binary batch files.\n"
102. "The CIFAR dataset could be downloaded at\n"
103. "    http://www.cs.toronto.edu/~kriz/cifar.html\n"
104. "You should gunzip them after downloading.\n");
105. } **else** {
106. google::InitGoogleLogging(argv[0]);
107. convert\_dataset(string(argv[1]), string(argv[2]), string(argv[3]));
108. //sh文件传递的参数：./build/examples/cifar10/convert\_cifar\_data.bin $DATA $EXAMPLE $DBTYPE ，依次为argv[0] argv[1] argv[2] argv[3]；
109. //即执行程序名称，原始数据存放位置,转换后数据保存的位置，转换的数据类型lmdb，以上参数都是以字符串形式进行传递的。
110. }
111. **return** 0;
112. }

db.cpp 文件

里面定义了caffe名字空间和其子空间db

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45367097)[在CODE上查看代码片](https://code.csdn.net/snippets/655717)

1. #include "caffe/util/db.hpp"
3. #include <sys/stat.h>
4. #include <string>
6. **namespace** caffe { **namespace** db {
8. **const** **size\_t** LMDB\_MAP\_SIZE = 1099511627776;  // 1 TB
10. //在制定位置以options方式创建（或打开）leveldb类型数据文件，并检查是否打开成功
11. **void** LevelDB::Open(**const** string& source, Mode mode) {
12. leveldb::Options options;//创建leveldb中的options类型对象
13. options.block\_size = 65536;
14. options.write\_buffer\_size = 268435456;
15. options.max\_open\_files = 100;
16. options.error\_if\_exists = mode == NEW;//mode=NEW时，是创建新leveldb类型文件，所以如果该文件以存在则报错
17. options.create\_if\_missing = mode != READ;//
18. leveldb::Status status = leveldb::DB::Open(options, source, &db\_);//通过leveldb空间中的DB子空间中的Open函数来创建（或打开）leveldb类型文件
19. CHECK(status.ok()) << "Failed to open leveldb " << source
20. << std::endl << status.ToString();
21. LOG(INFO) << "Opened leveldb " << source;
22. }

25. //Open函数主要负责，创建环境；设置环境参数，打开环境
26. **void** LMDB::Open(**const** string& source, Mode mode) {
27. MDB\_CHECK(mdb\_env\_create(&mdb\_env\_));//创建lmdb操作环境
28. MDB\_CHECK(mdb\_env\_set\_mapsize(mdb\_env\_, LMDB\_MAP\_SIZE));//设置环境内训映射
29. **if** (mode == NEW) {
30. CHECK\_EQ(mkdir(source.c\_str(), 0744), 0) << "mkdir " << source << "failed";
31. }//检查文件
32. **int** flags = 0;
33. **if** (mode == READ) {
34. flags = MDB\_RDONLY | MDB\_NOTLS;
35. }
36. MDB\_CHECK(mdb\_env\_open(mdb\_env\_, source.c\_str(), flags, 0664));//打开创建的环境
37. LOG(INFO) << "Opened lmdb " << source;
38. }
40. LMDBCursor\* LMDB::NewCursor() {
41. MDB\_txn\* mdb\_txn;
42. MDB\_cursor\* mdb\_cursor;
43. MDB\_CHECK(mdb\_txn\_begin(mdb\_env\_, NULL, MDB\_RDONLY, &mdb\_txn));
44. MDB\_CHECK(mdb\_dbi\_open(mdb\_txn, NULL, 0, &mdb\_dbi\_));
45. MDB\_CHECK(mdb\_cursor\_open(mdb\_txn, mdb\_dbi\_, &mdb\_cursor));
46. **return** **new** LMDBCursor(mdb\_txn, mdb\_cursor);
47. }
49. //在lmdb环境中创建操作句柄
50. LMDBTransaction\* LMDB::NewTransaction() {
51. MDB\_txn\* mdb\_txn;
52. MDB\_CHECK(mdb\_txn\_begin(mdb\_env\_, NULL, 0, &mdb\_txn));//创建操作句柄
53. MDB\_CHECK(mdb\_dbi\_open(mdb\_txn, NULL, 0, &mdb\_dbi\_));//打开数据库环境
54. **return** **new** LMDBTransaction(&mdb\_dbi\_, mdb\_txn);
55. }
57. **void** LMDBTransaction::Put(**const** string& key, **const** string& value) {
58. MDB\_val mdb\_key, mdb\_value;
59. mdb\_key.mv\_data = **const\_cast**<**char**\*>(key.data());
60. mdb\_key.mv\_size = key.size();
61. mdb\_value.mv\_data = **const\_cast**<**char**\*>(value.data());
62. mdb\_value.mv\_size = value.size();
63. MDB\_CHECK(mdb\_put(mdb\_txn\_, \*mdb\_dbi\_, &mdb\_key, &mdb\_value, 0));
64. }
66. DB\* GetDB(DataParameter::DB backend) {
67. **switch** (backend) {
68. **case** DataParameter\_DB\_LEVELDB:
69. **return** **new** LevelDB();
70. **case** DataParameter\_DB\_LMDB:
71. **return** **new** LMDB();
72. **default**:
73. LOG(FATAL) << "Unknown database backend";
74. }
75. }
77. //创建cafe::db“命名空间”类型对象，cafe::db“命名空间”中包含了各种数据操作函数
78. DB\* GetDB(**const** string& backend) {
79. **if** (backend == "leveldb") {
80. **return** **new** LevelDB();
81. } **else** **if** (backend == "lmdb") {
82. **return** **new** LMDB();
83. } **else** {
84. LOG(FATAL) << "Unknown database backend";
85. }
86. }
88. }  // namespace db
89. }  // namespace caffe

五，以上代码注释为个人理解，如有遗漏，错误还望大家多多交流，指正，以便共同学习，进步！！

[Caffe3——ImageNet数据集创建lmdb类型的数据](http://blog.csdn.net/whiteinblue/article/details/45485559)

2015-05-04 21:01 762人阅读 [评论](http://blog.csdn.net/whiteinblue/article/details/45485559#comments)(8) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/whiteinblue/article/details/45485559#report)

[caffe](http://www.csdn.net/tag/caffe)[imagetnet](http://www.csdn.net/tag/imagetnet)[lmdb数据转换](http://www.csdn.net/tag/lmdb%e6%95%b0%e6%8d%ae%e8%bd%ac%e6%8d%a2)

目录[(?)[+]](http://blog.csdn.net/whiteinblue/article/details/45485559)

ImageNet数据集和cifar，mnist数据集最大的不同，就是数据量特别大；单张图片尺寸大，训练样本个数多；面对如此大的数据集，在转换成lmdb文件时；使用了很多新的类型对象。

1，动态扩容的数组“vector”，动态地添加新元素

2，pair类型数据对，用于存储成对的对象，例如存储文件名和对应标签

3，利用opencv中的图像处理函数，来读取和处理大尺寸图像

**一：程序开始**

由于要向imageNet数据集中设置resize和是否乱序等参数，所以本文使用gflags命令行解析工具；在Create.sh文件中，调用convert\_imageset.bin语句为：

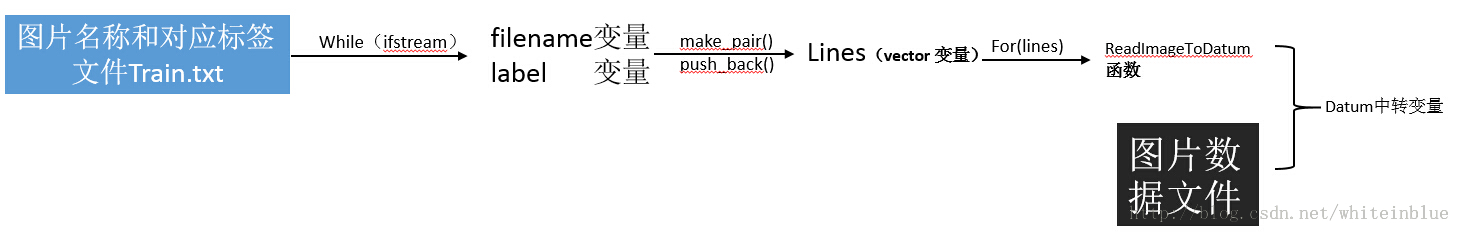
**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45485559)

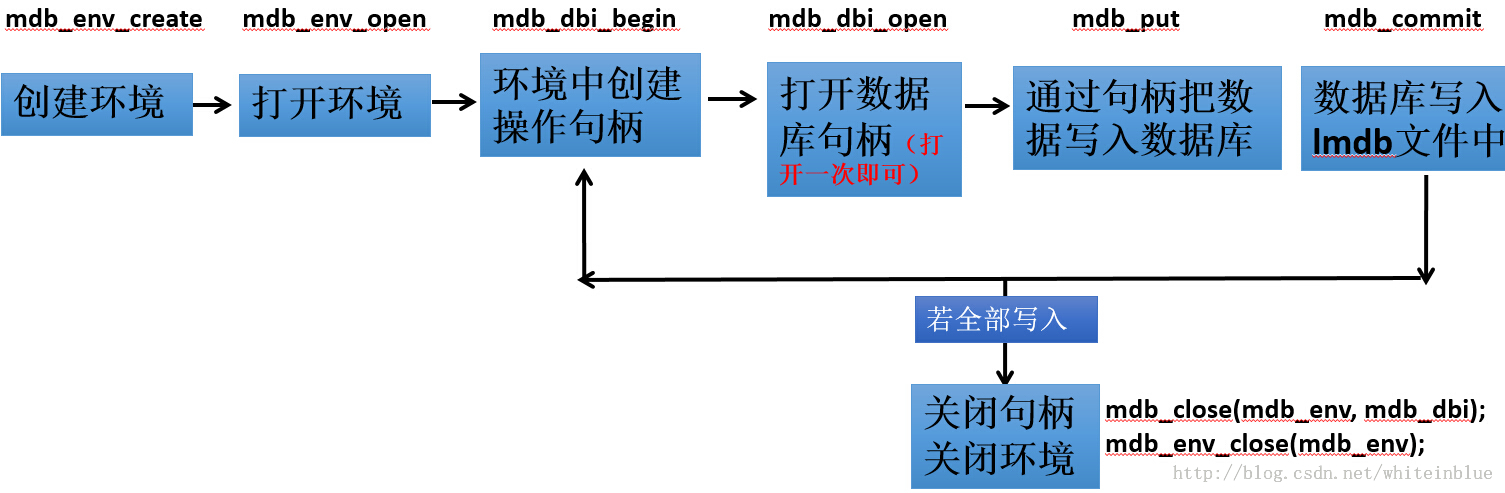
1. <pre name="code" **class**="cpp">GLOG\_logtostderr=1$TOOLS/convert\_imageset \
2. --resize\_height=$RESIZE\_HEIGHT \
3. --resize\_width=$RESIZE\_WIDTH \
4. --shuffle \
5. $TRAIN\_DATA\_ROOT \  图像数据集存放的根目录
6. $DATA/train.txt \       图像的ID和对应的分类标签数字
7. $EXAMPLE/ilsvrc12\_train\_lmdb  lmdb文件保存的路径

由于train.txt文件太大，电脑打不开，故打开val.txt一窥之；val.txt中的某个数据为：

65ILSVRC2012\_val\_00000002.JPEG ，65应该是对应的标签，后面的是图像的编号id。

**二：数据转换流程图**





**三：convert\_imageset.cpp函数分析**

**1引入必要的头文件和命名空间**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45485559)[在CODE上查看代码片](https://code.csdn.net/snippets/658716)

1. #include<algorithm>//输出数组的内容、对数组进行升幂排序、反转数组内容、复制数组内容等操作，
2. #include <fstream>  // NOLINT(readability/streams)
3. #include <string>
4. #include<utility>//utility头文件定义了一个pair类型,pair类型用于存储一对数据
5. #include<vector>//会自动扩展容量的数组
6. #include "boost/scoped\_ptr.hpp"//智能指针头文件
7. #include "gflags/gflags.h"
8. #include "glog/logging.h"
9. #include"caffe/proto/caffe.pb.h"
10. #include "caffe/util/db.hpp" //引入包装好的lmdb操作函数
11. #include "caffe/util/io.hpp" //引入opencv中的图像操作函数
12. #include "caffe/util/rng.hpp"

头文件和convert\_cifar\_data.cpp的区别：

1，引入gflags命令行解析工具；

2，引入utility头文件，里面提供了数组洗牌等操作

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45485559)[在CODE上查看代码片](https://code.csdn.net/snippets/658716)

1. **using** **namespace** caffe;  // NOLINT(build/namespaces)
2. **using** std::pair;
3. **using** boost::scoped\_ptr;

命名空间区别：

1，引入全部caffe命名空间

2，引入pair对命名空间

**2 gflags宏定义参数**

//通过gflags宏定义一些程序的参数变量

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45485559)[在CODE上查看代码片](https://code.csdn.net/snippets/658716)

1. DEFINE\_bool(gray, **false**,"When thisoption is on, treat images as grayscale ones");//是否为灰度图片
2. DEFINE\_bool(shuffle, **false**,"Randomlyshuffle the order of images and their labels");//定义洗牌变量，是否随机打乱数据集的顺序
3. DEFINE\_string(backend, "lmdb","The backend {lmdb, leveldb} for storing the result");//默认转换的数据类型
4. DEFINE\_int32(resize\_width, 0, "Width images areresized to");//定义resize的尺寸，默认为0，不转换尺寸
5. DEFINE\_int32(resize\_height, 0, "Height imagesare resized to");
6. DEFINE\_bool(check\_size, **false**,"When this optionis on, check that all the datum have the samesize");
7. DEFINE\_bool(encoded, **false**,"When this option ison, the encoded image will be save in datum");//用于转换数据格式的
8. DEFINE\_string(encode\_type, "","Optional:What type should we encode the image as ('png','jpg',...).");//要转换的数据格式

**3 main（）函数**

没有想cifar和mnist的main函数，通过调用convert\_data()函数来转换数据，而是直接在main函数内完成了所有数据转换代码。

**3.1 通过gflags宏定义接收命令行中传入的参数**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45485559)[在CODE上查看代码片](https://code.csdn.net/snippets/658716)

1. **const** boolis\_color = !FLAGS\_gray;  //通过gflags把宏定义变量的值，赋值给常值变量
2. **const** boolcheck\_size = FLAGS\_check\_size; //检查图像的size
3. **const** boolencoded = FLAGS\_encoded;//是否编译（转换）图像格式
4. **const** stringencode\_type = FLAGS\_encode\_type;//要编译的图像格式

**3.2读取源数据**

**3.2.1创建读取对象变量**

std::ifstream infile(argv[2]);//创建指向train.txt文件的文件读入流

std::vector<std::pair<std::string, int> > lines;//定义向量变量，向量中每个元素为一个pair对，pair对有两个成员变量，一个为string类型，一个为int类型；其中string类型用于存储文件名，int类型，感觉用于存数对应类别的id

如val.txt中前几个字符为“ILSVRC2012\_val\_00000001.JPEG65ILSVRC2012\_val\_00000002.JPEG”；感觉这个string= ILSVRC2012\_val\_00000001.JPEG   int=65

std::stringfilename;

int label;

**3.2.2 读取数据**

//下面一条while语句是把train.txt文件中存放的所有文件名和标签，都存放到vextor类型变量lines中；lines中存放图片的名字和对应的标签，不存储真正的图片数据

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45485559)[在CODE上查看代码片](https://code.csdn.net/snippets/658716)

1. **while** (infile>> filename >> label) {
2. nes.push\_back(std::make\_pair(filename, label));

//make\_pair是pair模板中定义的给pair对象赋值的函数，push\_back（）函数是vector对象的一个成员函数，用来在末端添加新元素}

**3.3判断是否进行洗牌操作**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45485559)[在CODE上查看代码片](https://code.csdn.net/snippets/658716)

1. **if**(FLAGS\_shuffle) {
2. // randomlyshuffle data
3. LOG(INFO)<< "Shuffling data";

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45485559)[在CODE上查看代码片](https://code.csdn.net/snippets/658716)

1. <span style="font-family: Arial, Helvetica, sans-serif; background-color: rgb(255, 255, 255);">//洗牌函数，使用随机生成器g对元素[first,last)容器内部元素进行随机排列</span>

 shuffle(lines.begin(), lines.end());//vector.begin() - 回传一个Iterator迭代器，它指向 vector 第一个元素。}

**3.4以智能指针的方式创建db::DB类型的对象 db**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45485559)[在CODE上查看代码片](https://code.csdn.net/snippets/658716)

1. scoped\_ptr<db::DB>db(db::GetDB(FLAGS\_backend));
2. //智能指针的创建方式类似泛型的格式，上面通过db.cpp内定义的命名的子命名空间中db的“成员函数”GetDB函数来初始化db对象
3. db->Open(argv[3], db::NEW);//argv[3]的文件夹下创建并打开lmdb的操作环境
4. scoped\_ptr<db::Transaction>txn(db->NewTransaction());//创建lmdb文件的操作句柄

**3.5 源数据中提取图像数据**

**3.5.1 通过ReadImageToDatum函数把图像数据读取到datum中**

//到源数据位置读取每张图片的数据。（../imagenet/xxx.jpeg,65,256,256,true,jpeg,&datum）

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45485559)[在CODE上查看代码片](https://code.csdn.net/snippets/658716)

1. status= ReadImageToDatum(root\_folder + lines[line\_id].first，lines[line\_id].second, resize\_height,resize\_width, is\_color,enc, &datum); //把图像数据读取到datum中

**3.5.2  ReadImageToDatum函数说明**

ReadImageToDatum函数为io.cpp文件中定义的函数；io.cpp主要实现了3部分功能：

1，从text文件或者二进制文件中读写proto文件；

2，利用opencv的Mat矩阵，把图像数据读到Mat矩阵中；

3，把Mat矩阵中的值放入到datum中

**3.5.3 检查数据尺寸**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45485559)[在CODE上查看代码片](https://code.csdn.net/snippets/658716)

1. **if** (check\_size) {//检查图片尺寸
2. **if** (!data\_size\_initialized) {//若data\_size\_initialized没有初始化
3. data\_size = datum.channels() \*datum.height() \* datum.width();
4. data\_size\_initialized = **true**;
5. } **else** {
6. **const** std::string& data =datum.data();
7. CHECK\_EQ(data.size(), data\_size)<< "Incorrect data field size "<< data.size();
8. }

**3.6   序列化键和值并放入临时数据库**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45485559)[在CODE上查看代码片](https://code.csdn.net/snippets/658716)

1. // sequential
2. intlength = snprintf(key\_cstr, kMaxKeyLength, "%08d\_%s", line\_id,lines[line\_id].first.c\_str());//若line\_id=1234，lines[line\_id].first=“abc.jpeg” 则 key\_str=00001234\_abc.jpeg，length=00001234\_abc.jpeg字符串的长度
3. // Put in db
4. string out;
5. CHECK(datum.SerializeToString(&out));//datum数据，序列化到字符串中
6. txn->Put(string(key\_cstr, length), out);//把键值对放入到数据库

**3.7 批量提交到lmdb文件**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45485559)[在CODE上查看代码片](https://code.csdn.net/snippets/658716)

1. **if** (++count % 1000 == 0) {
2. // Commit db
3. txn->Commit();//保存到lmdb类型的文件
4. txn.reset(db->NewTransaction());//重新初始化操作句柄
5. LOG(ERROR) << "Processed" << count << " files.";
6. }

**四，相关文件**

**4.1 Convert\_imageset.cpp文件**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45485559)[在CODE上查看代码片](https://code.csdn.net/snippets/658716)

1. // This program converts a set of images to a lmdb/leveldb by storing them
2. // as Datum proto buffers.
3. // Usage:
4. //   convert\_imageset [FLAGS] ROOTFOLDER/ LISTFILE DB\_NAME
5. //
6. // where ROOTFOLDER is the root folder that holds all the images, and LISTFILE
7. // should be a list of files as well as their labels, in the format as
8. //   subfolder1/file1.JPEG 7
9. //   ....
11. #include <algorithm>//输出数组的内容、对数组进行升幂排序、反转数组内容、复制数组内容等操作，
12. #include <fstream>  // NOLINT(readability/streams)
13. #include <string>
14. #include <utility>//utility头文件定义了一个pair类型
15. #include <vector>//会自动扩展容量的数组
17. #include "boost/scoped\_ptr.hpp"
18. #include "gflags/gflags.h"
19. #include "glog/logging.h"
21. #include "caffe/proto/caffe.pb.h"
22. #include "caffe/util/db.hpp"
23. #include "caffe/util/io.hpp"
24. #include "caffe/util/rng.hpp"
26. **using** **namespace** caffe;  // NOLINT(build/namespaces)
27. **using** std::pair;
28. **using** boost::scoped\_ptr;

31. //通过gflags宏定义一些程序的参数变量
32. DEFINE\_bool(gray, **false**,
33. "When this option is on, treat images as grayscale ones");
34. DEFINE\_bool(shuffle, **false**,
35. "Randomly shuffle the order of images and their labels");//洗牌，随机打乱数据集的顺序
36. DEFINE\_string(backend, "lmdb",
37. "The backend {lmdb, leveldb} for storing the result");
38. DEFINE\_int32(resize\_width, 0, "Width images are resized to");
39. DEFINE\_int32(resize\_height, 0, "Height images are resized to");
40. DEFINE\_bool(check\_size, **false**,
41. "When this option is on, check that all the datum have the same size");
42. DEFINE\_bool(encoded, **false**,
43. "When this option is on, the encoded image will be save in datum");//用于转换数据格式的
44. DEFINE\_string(encode\_type, "",
45. "Optional: What type should we encode the image as ('png','jpg',...).");//要转换的数据格式
47. **int** main(**int** argc, **char**\*\* argv) {
48. ::google::InitGoogleLogging(argv[0]);
50. #ifndef GFLAGS\_GFLAGS\_H\_
51. **namespace** gflags = google;
52. #endif
54. gflags::SetUsageMessage("Convert a set of images to the leveldb/lmdb\n"
55. "format used as input for Caffe.\n"
56. "Usage:\n"
57. "    convert\_imageset [FLAGS] ROOTFOLDER/ LISTFILE DB\_NAME\n"
58. "The ImageNet dataset for the training demo is at\n"
59. "    http://www.image-net.org/download-images\n");
60. gflags::ParseCommandLineFlags(&argc, &argv, **true**);
62. **if** (argc < 4) {
63. gflags::ShowUsageWithFlagsRestrict(argv[0], "tools/convert\_imageset");
64. **return** 1;
65. }
66. //arg[1] 训练集存放的地址，arg[2] train.txt（估计是训练集中所有图片的文件名称），arg[3] 要保存的文件名称xxlmdb
67. **const** **bool** is\_color = !FLAGS\_gray;  //通过gflags把宏定义变量的值，赋值给常值变量
68. **const** **bool** check\_size = FLAGS\_check\_size; //检查图像的size
69. **const** **bool** encoded = FLAGS\_encoded;//是否编译（转换）图像格式
70. **const** string encode\_type = FLAGS\_encode\_type;//要编译的图像格式
72. std::ifstream infile(argv[2]);//定义指向train.txt数据文件的文件读入流
73. std::vector<std::pair<std::string, **int**> > lines;//定义向量变量，向量中每个元素为一个pair对，pair对有两个成员变量，一个为string类型，一个为int类型
74. std::string filename;
75. **int** label;
76. //下面一条while语句是把train.txt文件中存数的数据和标签，都存放到vextor类型变量中lines中；lines中存放图片的名字和对应的标签，不存储真正的图片数据
77. **while** (infile >> filename >> label) {
78. lines.push\_back(std::make\_pair(filename, label));//make\_pair是pair模板中定义的给pair对象赋值的函数，push\_back（）函数是vector对象的一个成员函数，用来在末端添加新元素
79. }
80. **if** (FLAGS\_shuffle) {
81. // randomly shuffle data
82. LOG(INFO) << "Shuffling data";
83. //洗牌函数，使用随机生成器g对元素[first, last)容器内部元素进行随机排列
84. shuffle(lines.begin(), lines.end());//vector.begin() - 回传一个Iterator迭代器，它指向 vector 第一个元素。
85. }
86. LOG(INFO) << "A total of " << lines.size() << " images.";
88. **if** (encode\_type.size() && !encoded)
89. LOG(INFO) << "encode\_type specified, assuming encoded=true.";
91. **int** resize\_height = std::max<**int**>(0, FLAGS\_resize\_height);
92. **int** resize\_width = std::max<**int**>(0, FLAGS\_resize\_width);
94. // Create new DB
95. scoped\_ptr<db::DB> db(db::GetDB(FLAGS\_backend));
96. db->Open(argv[3], db::NEW);//argv[3]的文件夹下打开创建lmdb的操作环境
97. scoped\_ptr<db::Transaction> txn(db->NewTransaction());//创建lmdb文件的操作句柄
99. // Storing to db
100. std::string root\_folder(argv[1]);//把源数据文件的地址复制给root\_folder
101. Datum datum;//声明数据“转换”对象
102. **int** count = 0;
103. **const** **int** kMaxKeyLength = 256;
104. **char** key\_cstr[kMaxKeyLength];
105. **int** data\_size = 0;
106. **bool** data\_size\_initialized = **false**;
108. **for** (**int** line\_id = 0; line\_id < lines.size(); ++line\_id) {
109. **bool** status;
110. std::string enc = encode\_type; //enc为空串，则enc.size()=false;否则为true
111. **if** (encoded && !enc.size()) {
112. // Guess the encoding type from the file name
113. string fn = lines[line\_id].first;//把图像的文件名赋值给fn（filename）
114. **size\_t** p = fn.rfind('.');//rfind函数的返回值是一个整形的索引值，直线要查找的字符在字符串中的位置；若没有找到，返回string::npos
115. **if** ( p == fn.npos )
116. LOG(WARNING) << "Failed to guess the encoding of '" << fn << "'";
117. enc = fn.substr(p);//找到了，就截取文件名”.“后面的字符串，以获得图像格式字符串
118. std::transform(enc.begin(), enc.end(), enc.begin(), ::tolower);//将enc字符串转换成小写
119. }
120. //到源数据位置，以此读取每张图片的数据。（../imagenet/xxx.jpeg,65,256,256,true,jpeg,&datum）
121. status = ReadImageToDatum(root\_folder + lines[line\_id].first,
122. lines[line\_id].second, resize\_height, resize\_width, is\_color,enc, &datum);  //把图像数据读取到datum中
123. **if** (status == **false**) **continue**;//status=false,说明此张图片读取错误；“跳过”继续下一张
124. **if** (check\_size) {//检查图片尺寸
125. **if** (!data\_size\_initialized) {//若data\_size\_initialized没有初始化
126. data\_size = datum.channels() \* datum.height() \* datum.width();
127. data\_size\_initialized = **true**;
128. } **else** {
129. **const** std::string& data = datum.data();
130. CHECK\_EQ(data.size(), data\_size) << "Incorrect data field size "
131. << data.size();
132. }
133. }
134. // sequential
135. **int** length = snprintf(key\_cstr, kMaxKeyLength, "%08d\_%s", line\_id,
136. lines[line\_id].first.c\_str());//若line\_id=1234，lines[line\_id].first=“abc.jpeg” 则 key\_str=00001234\_abc.jpeg，length=00001234\_abc.jpeg字符串的长度
138. // Put in db
139. string out;
140. CHECK(datum.SerializeToString(&out));//datum数据，序列化到字符串中
141. txn->Put(string(key\_cstr, length), out);//把键值对放入到数据库
143. **if** (++count % 1000 == 0) {
144. // Commit db
145. txn->Commit();//保存到lmdb类型的文件
146. txn.reset(db->NewTransaction());//重新初始化操作句柄
147. LOG(ERROR) << "Processed " << count << " files.";
148. }
149. }
150. // write the last batch
151. **if** (count % 1000 != 0) {
152. txn->Commit();
153. LOG(ERROR) << "Processed " << count << " files.";
154. }
155. **return** 0;
156. }

**4.2 io.cpp文件**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45485559)[在CODE上查看代码片](https://code.csdn.net/snippets/658716)

1. #include <fcntl.h>
2. #include <google/protobuf/io/coded\_stream.h>
3. #include <google/protobuf/io/zero\_copy\_stream\_impl.h>
4. #include <google/protobuf/text\_format.h>
5. #include <opencv2/core/core.hpp>
6. #include <opencv2/highgui/highgui.hpp>
7. #include <opencv2/highgui/highgui\_c.h>
8. #include <opencv2/imgproc/imgproc.hpp>
9. #include <stdint.h>
11. #include <algorithm>
12. #include <fstream>  // NOLINT(readability/streams)
13. #include <string>
14. #include <vector>
16. #include "caffe/common.hpp"
17. #include "caffe/proto/caffe.pb.h"
18. #include "caffe/util/io.hpp"
20. **const** **int** kProtoReadBytesLimit = INT\_MAX;  // Max size of 2 GB minus 1 byte.
22. **namespace** caffe {
24. **using** google::protobuf::io::FileInputStream;//文件输入流
25. **using** google::protobuf::io::FileOutputStream;//文件输出流
26. **using** google::protobuf::io::ZeroCopyInputStream;//These interfaces are different from classic I/O streams in that they try to minimize the amount of data copying that needs to be done
27. **using** google::protobuf::io::CodedInputStream;
28. **using** google::protobuf::io::ZeroCopyOutputStream;
29. **using** google::protobuf::io::CodedOutputStream;
30. **using** google::protobuf::Message;
32. **bool** ReadProtoFromTextFile(**const** **char**\* filename, Message\* proto) {//从文本文件中读入proto文件
33. **int** fd = open(filename, O\_RDONLY);
34. CHECK\_NE(fd, -1) << "File not found: " << filename;
35. FileInputStream\* input = **new** FileInputStream(fd);
36. **bool** success = google::protobuf::TextFormat::Parse(input, proto);
37. **delete** input;
38. close(fd);
39. **return** success;
40. }
42. **void** WriteProtoToTextFile(**const** Message& proto, **const** **char**\* filename) {//想文本文件中写入proto文件
43. **int** fd = open(filename, O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);
44. FileOutputStream\* output = **new** FileOutputStream(fd);
45. CHECK(google::protobuf::TextFormat::Print(proto, output));
46. **delete** output;
47. close(fd);
48. }
50. **bool** ReadProtoFromBinaryFile(**const** **char**\* filename, Message\* proto) {//从二进制文件读入proto
51. **int** fd = open(filename, O\_RDONLY);
52. CHECK\_NE(fd, -1) << "File not found: " << filename;
53. ZeroCopyInputStream\* raw\_input = **new** FileInputStream(fd);
54. CodedInputStream\* coded\_input = **new** CodedInputStream(raw\_input);
55. coded\_input->SetTotalBytesLimit(kProtoReadBytesLimit, 536870912);
57. **bool** success = proto->ParseFromCodedStream(coded\_input);
59. **delete** coded\_input;
60. **delete** raw\_input;
61. close(fd);
62. **return** success;
63. }
65. **void** WriteProtoToBinaryFile(**const** Message& proto, **const** **char**\* filename) {//把proto写入二进制文件中
66. fstream output(filename, ios::out | ios::trunc | ios::binary);
67. CHECK(proto.SerializeToOstream(&output));
68. }

71. //基本上讲 Mat 是一个类，由两个数据部分组成：矩阵头（包含矩阵尺寸，存储方法，存储地址等信息）和
72. //一个指向存储所有像素值的矩阵的指针（根据所选存储方法的不同矩阵可以是不同的维数）。
73. //矩阵头的尺寸是常数值，但矩阵本身的尺寸会依图像的不同而不同，通常比矩阵头的尺寸大数个数量级。因此，当在程序中传递图像并创建拷贝时，
74. //大的开销是由矩阵造成的，而不是信息头。OpenCV是一个图像处理库，囊括了大量的图像处理函数，为了解决问题通常要使用库中的多个函数，
75. //因此在函数中传递图像是家常便饭。同时不要忘了我们正在讨论的是计算量很大的图像处理算法，因此，除非万不得已，我们不应该拷贝 大 的图像，因为这会降低程序速度。
76. cv::Mat ReadImageToCVMat(**const** string& filename,
77. **const** **int** height, **const** **int** width, **const** **bool** is\_color) {//读取图片到CVMat中，cv::Mat ，Mat数据结构式opencv2.0以后的特定的数据类型
78. cv::Mat cv\_img;
79. **int** cv\_read\_flag = (is\_color ? CV\_LOAD\_IMAGE\_COLOR :
80. CV\_LOAD\_IMAGE\_GRAYSCALE);
81. cv::Mat cv\_img\_origin = cv::imread(filename, cv\_read\_flag);//读取图片内容
82. **if** (!cv\_img\_origin.data) {
83. LOG(ERROR) << "Could not open or find file " << filename;
84. **return** cv\_img\_origin;
85. }
86. **if** (height > 0 && width > 0) {
87. cv::resize(cv\_img\_origin, cv\_img, cv::Size(width, height));
88. } **else** {
89. cv\_img = cv\_img\_origin;
90. }
91. **return** cv\_img;
92. }
94. cv::Mat ReadImageToCVMat(**const** string& filename,//读取图片到CVMat中，重载1
95. **const** **int** height, **const** **int** width) {
96. **return** ReadImageToCVMat(filename, height, width, **true**);
97. }
99. cv::Mat ReadImageToCVMat(**const** string& filename,//读取图片到CVMat中，重载2
100. **const** **bool** is\_color) {
101. **return** ReadImageToCVMat(filename, 0, 0, is\_color);
102. }
104. cv::Mat ReadImageToCVMat(**const** string& filename) {//读取图片到CVMat中，重载3
105. **return** ReadImageToCVMat(filename, 0, 0, **true**);
106. }
107. // Do the file extension and encoding match?
108. **static** **bool** matchExt(**const** std::string & fn, //匹配拓展名称？
109. std::string en) {
110. **size\_t** p = fn.rfind('.');//查找"."字符，若找到则返回“.”在字符串中的位置，找不到则返回npos
111. std::string ext = p != fn.npos ? fn.substr(p) : fn;//如果字符串fn中存在".“，则截取字符串p
112. std::transform(ext.begin(), ext.end(), ext.begin(), ::tolower);//把ext变成小写
113. std::transform(en.begin(), en.end(), en.begin(), ::tolower);
114. **if** ( ext == en )
115. **return** **true**;
116. **if** ( en == "jpg" && ext == "jpeg" )
117. **return** **true**;
118. **return** **false**;
119. }
121. **bool** ReadImageToDatum(**const** string& filename, **const** **int** label,//把图片读到 Datum中
122. **const** **int** height, **const** **int** width, **const** **bool** is\_color,
123. **const** std::string & encoding, Datum\* datum) {
124. cv::Mat cv\_img = ReadImageToCVMat(filename, height, width, is\_color);//先把数据读到cv::Mat类型矩阵中
125. **if** (cv\_img.data) {//Mat矩阵中数据指针Mat.data是uchar类型指针,矩阵中的元素应该是uchar类型；该语句是判断cv\_img中是否有数据
126. **if** (encoding.size()) {//是否需要编码
127. **if** ( (cv\_img.channels() == 3) == is\_color && !height && !width &&
128. matchExt(filename, encoding) )
129. **return** ReadFileToDatum(filename, label, datum);
131. std::vector<uchar> buf;
132. cv::imencode("."+encoding, cv\_img, buf);//感觉这行代码的作用是把cv\_img中的值赋值给buf
133. datum->set\_data(std::string(**reinterpret\_cast**<**char**\*>(&buf[0]),
134. buf.size()));
135. datum->set\_label(label);
136. datum->set\_encoded(**true**);//感觉是一种编码函数
137. **return** **true**;
138. }
139. CVMatToDatum(cv\_img, datum);
140. datum->set\_label(label);
141. **return** **true**;
142. } **else** {
143. **return** **false**;
144. }
145. }
147. **bool** ReadFileToDatum(**const** string& filename, **const** **int** label,
148. Datum\* datum) {
149. std::streampos size;
151. fstream file(filename.c\_str(), ios::in|ios::binary|ios::ate);
152. **if** (file.is\_open()) {
153. size = file.tellg();
154. std::string buffer(size, ' ');
155. file.seekg(0, ios::beg);
156. file.read(&buffer[0], size);
157. file.close();
158. datum->set\_data(buffer);
159. datum->set\_label(label);
160. datum->set\_encoded(**true**);
161. **return** **true**;
162. } **else** {
163. **return** **false**;
164. }
165. }
167. cv::Mat DecodeDatumToCVMatNative(**const** Datum& datum) {
168. cv::Mat cv\_img;
169. CHECK(datum.encoded()) << "Datum not encoded";
170. **const** string& data = datum.data();
171. std::vector<**char**> vec\_data(data.c\_str(), data.c\_str() + data.size());
172. cv\_img = cv::imdecode(vec\_data, -1);
173. **if** (!cv\_img.data) {
174. LOG(ERROR) << "Could not decode datum ";
175. }
176. **return** cv\_img;
177. }
178. cv::Mat DecodeDatumToCVMat(**const** Datum& datum, **bool** is\_color) {
179. cv::Mat cv\_img;
180. CHECK(datum.encoded()) << "Datum not encoded";
181. **const** string& data = datum.data();
182. std::vector<**char**> vec\_data(data.c\_str(), data.c\_str() + data.size());
183. **int** cv\_read\_flag = (is\_color ? CV\_LOAD\_IMAGE\_COLOR :
184. CV\_LOAD\_IMAGE\_GRAYSCALE);
185. cv\_img = cv::imdecode(vec\_data, cv\_read\_flag);
186. **if** (!cv\_img.data) {
187. LOG(ERROR) << "Could not decode datum ";
188. }
189. **return** cv\_img;
190. }
192. // If Datum is encoded will decoded using DecodeDatumToCVMat and CVMatToDatum
193. // If Datum is not encoded will do nothing
194. **bool** DecodeDatumNative(Datum\* datum) {
195. **if** (datum->encoded()) {
196. cv::Mat cv\_img = DecodeDatumToCVMatNative((\*datum));
197. CVMatToDatum(cv\_img, datum);
198. **return** **true**;
199. } **else** {
200. **return** **false**;
201. }
202. }
203. **bool** DecodeDatum(Datum\* datum, **bool** is\_color) {
204. **if** (datum->encoded()) {
205. cv::Mat cv\_img = DecodeDatumToCVMat((\*datum), is\_color);
206. CVMatToDatum(cv\_img, datum);
207. **return** **true**;
208. } **else** {
209. **return** **false**;
210. }
211. }
213. **void** CVMatToDatum(**const** cv::Mat& cv\_img, Datum\* datum) {
214. CHECK(cv\_img.depth() == CV\_8U) << "Image data type must be unsigned byte";
215. datum->set\_channels(cv\_img.channels());
216. datum->set\_height(cv\_img.rows);
217. datum->set\_width(cv\_img.cols);
218. datum->clear\_data();
219. datum->clear\_float\_data();
220. datum->set\_encoded(**false**);
221. **int** datum\_channels = datum->channels();
222. **int** datum\_height = datum->height();
223. **int** datum\_width = datum->width();
224. **int** datum\_size = datum\_channels \* datum\_height \* datum\_width;
225. std::string buffer(datum\_size, ' ');
226. **for** (**int** h = 0; h < datum\_height; ++h) {
227. **const** uchar\* ptr = cv\_img.ptr<uchar>(h);
228. **int** img\_index = 0;
229. **for** (**int** w = 0; w < datum\_width; ++w) {
230. **for** (**int** c = 0; c < datum\_channels; ++c) {
231. **int** datum\_index = (c \* datum\_height + h) \* datum\_width + w;
232. buffer[datum\_index] = **static\_cast**<**char**>(ptr[img\_index++]);
233. }
234. }
235. }
236. datum->set\_data(buffer);
237. }
238. 。。。。。

五，以上代码注释为个人理解，如有遗漏，错误还望大家多多交流，指正，以便共同学习，进步！！

[Caffe4——计算图像均值](http://blog.csdn.net/whiteinblue/article/details/45540301)

2015-05-06 20:17 273人阅读 [评论](http://blog.csdn.net/whiteinblue/article/details/45540301#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/whiteinblue/article/details/45540301#report)

[caffe源码解析](http://www.csdn.net/tag/caffe%e6%ba%90%e7%a0%81%e8%a7%a3%e6%9e%90)[计算图像均值](http://www.csdn.net/tag/%e8%ae%a1%e7%ae%97%e5%9b%be%e5%83%8f%e5%9d%87%e5%80%bc)

目录[(?)[+]](http://blog.csdn.net/whiteinblue/article/details/45540301)

均值削减是数据预处理中常见的处理方式，按照之前在学习ufldl教程PCA的一章时，对于图像介绍了两种：第一种常用的方式叫做dimension\_mean（个人命名），是依据输入数据的维度，每个维度内进行削减，这个也是常见的做法；第二种叫做per\_image\_mean，ufldl教程上说，在natural images上训练网络时；给每个像素（这里只每个dimension）计算一个独立的均值和方差是make little sense的；这是因为图像本身具有统计不变性，即在图像的一部分的统计特性和另一部分相同。作者最后建议，如果你训练你的算法在非natural images（如mnist，或者在白背景存在单个独立的物体），其他类型的规则化是值得考虑的。但是当在natural images上训练时，per\_image\_mean是一个合理的默认选择。

本文中在imagenet数据集上采用的是dimension\_mean的方法。

**一：程序开始**

make\_image\_mean.sh文件调用代码：

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45540301)[在CODE上查看代码片](https://code.csdn.net/snippets/660531)

1. EXAMPLE=examples/imagenet
2. DATA=data/ilsvrc12
3. TOOLS=build/tools
4. $TOOLS/compute\_image\_mean $EXAMPLE/ilsvrc12\_train\_lmdb \
5. $DATA/imagenet\_mean.binaryproto<strong>
6. </strong>

**二：make\_image\_mean.cpp函数分析**

输入参数：lmdb文件 均值文件imagenet\_mean.binaryproto

**2.1 头文件分析**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45540301)[在CODE上查看代码片](https://code.csdn.net/snippets/660531)

1. #include<stdint.h>//定义了几种扩展的整数类型和宏
2. #include<algorithm>//输出数组的内容、对数组进行排序、反转数组内容、复制数组内容等操作，
3. #include<string>
4. #include<utility>//utility头文件定义了一个pair类型,pair类型用于存储一对数据;它也提供一些常用的便利函数、或类、或模板。大小求值、值交换：min、max和swap。
5. #include<vector>//可以自动扩展容量的数组
7. #include"boost/scoped\_ptr.hpp"
8. #include"gflags/gflags.h"
9. #include"glog/logging.h"
11. #include"caffe/proto/caffe.pb.h"
12. #include"caffe/util/db.hpp"//引入包装好的lmdb操作函数
13. #include"caffe/util/io.hpp"//引入opencv中的图像操作函数
14. usingnamespacecaffe;  //引入caffe命名空间
15. usingstd::max;//
16. usingstd::pair;
17. **using** boost::scoped\_ptr;

**2.2 gflags宏定义string变量**

DEFINE\_string(backend, "lmdb","The backend {leveldb, lmdb} containing theimages");

**2.3 main函数分析**

**2.3.1 lmdb数据操作**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45540301)[在CODE上查看代码片](https://code.csdn.net/snippets/660531)

1. scoped\_ptr<db::DB>db(db::GetDB(FLAGS\_backend));
2. db->Open(argv[1], db::READ);//只读的方式打开lmdb文件
3. scoped\_ptr<db::Cursor> cursor(db->NewCursor());
4. //lmdb数据库的“光标”文件，一个光标保存一个从数据库根目录到数据库文件的路径；A cursorholds a path of (page pointer, key index) from the DB root to a position in theDB, plus other state.

**2.3.4 声明中转对象变量**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45540301)[在CODE上查看代码片](https://code.csdn.net/snippets/660531)

1. BlobProtosum\_blob;//声明blob变量；这个BlobProto在哪里定义的，没有找到；感觉应该在caffe.pb.h中定义的，因为db.cpp和io.cpp中没有找到
2. **int** count = 0;
3. // load first datum
4. Datum datum;
5. datum.ParseFromString(cursor->value());//这个cursor.value，感觉返回的应该是lmdb中存储的第一个键值对数据

**2.3.5 给BlobProto类型变量赋值**

每个blob对象，为一个4维的数组，分别为image\_num\*channels\*height\*width

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45540301)[在CODE上查看代码片](https://code.csdn.net/snippets/660531)

1. sum\_blob.set\_num(1);//设置图片的个数
2. sum\_blob.set\_channels(datum.channels());
3. sum\_blob.set\_height(datum.height());
4. sum\_blob.set\_width(datum.width());
5. constintdata\_size = datum.channels() \*datum.height() \* datum.width();//每张图片的尺寸
6. intsize\_in\_datum = std::max<**int**>(datum.data().size(),datum.float\_data\_size());

这个size（）和float\_data\_size（）有些不明白，图像数据正常应该是整形的数据（例如uint8\_t），感觉这个size()应该对应的是整型数据的个数，例如一个50\*50的彩色图片，最后应该是50\*50\*3=750个整型数来表示一幅50\*50的图片；至于这个float\_data\_size()就不清楚了，感觉是某些图片数据使用float类型存储的，所以用float来统计数值的个数。开始感觉这个float的size应该是把int类型转换成float后，查看在float类型下的字节占用情况；但是由下面的代码来看，感觉这个size(),统计的是数据的个数也就是750，而不是占用的字节数。如果图像使用int类型存储的，那么float\_data\_size()=0；如果使用float类型存储的，那么datum.data.size=0。所以每次都要max操作

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45540301)[在CODE上查看代码片](https://code.csdn.net/snippets/660531)

1. **for** (inti= 0; i<size\_in\_datum; ++i) {
2. sum\_blob.add\_data(0.);//设置初值为float型的0.0
3. }

**2.3.6利用循环和cursor读取lmdb中的数据**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45540301)[在CODE上查看代码片](https://code.csdn.net/snippets/660531)

1. **while** (cursor->valid()) {//如果cursor是有效的
2. Datum datum;
3. datum.ParseFromString(cursor->value());//解析cuisor.value返回的字符串值，到datum
4. DecodeDatumNative(&datum);//感觉是把datum中字符串类型的值，变成相应的类型
5. conststd::string& data =datum.data();//利用data来引用datum.data
6. size\_in\_datum = std::max<**int**>(datum.data().size(),datum.float\_data\_size());
7. CHECK\_EQ(size\_in\_datum,data\_size) <<"Incorrect data field size"<<size\_in\_datum;
8. **if** (data.size() != 0) {//datum.data().size()!=0
9. CHECK\_EQ(data.size(),size\_in\_datum);//判断是否相等
10. **for** (inti= 0; i<size\_in\_datum; ++i) {
11. sum\_blob.set\_data(i, sum\_blob.data(i) + (uint8\_t)data[i]);//对应位置的像素值相加（uin8\_t类型相加），相加的结果放在sum\_blob中
12. }
13. } **else**{
14. CHECK\_EQ(datum.float\_data\_size(), size\_in\_datum);
15. **for** (inti= 0; i<size\_in\_datum; ++i) {
16. sum\_blob.set\_data(i, sum\_blob.data(i) +
17. **static\_cast**<**float**>(datum.float\_data(i)));//对应位置的像素值相加（float类型相加）
18. }
19. }
20. ++count;
21. **if** (count % 10000 == 0) {
22. LOG(INFO) <<"Processed "<<count <<" files.";
23. }
24. cursor->Next();//光标下移（指针），指向下一个存储在lmdb中的数据
25. }

**2.3.7 求均值**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45540301)[在CODE上查看代码片](https://code.csdn.net/snippets/660531)

1. **for** (inti= 0; i<sum\_blob.data\_size(); ++i) {
2. sum\_blob.set\_data(i, sum\_blob.data(i) / count);
3. }

**2.3.8 存储到指定文件**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45540301)[在CODE上查看代码片](https://code.csdn.net/snippets/660531)

1. // Write to disk
2. **if** (argc == 3) {
3. LOG(INFO) <<"Write to "<<argv[2];
4. WriteProtoToBinaryFile(sum\_blob, argv[2]);
5. }

**2.3.9 计算每个channel的均值，这个貌似没有用到吧！**

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45540301)[在CODE上查看代码片](https://code.csdn.net/snippets/660531)

1. constint channels = sum\_blob.channels();
2. constint dim = sum\_blob.height() \*sum\_blob.width();
3. std::vector<**float**>mean\_values(channels,0.0);//容量为3的数组，初始值为0.0
4. LOG(INFO) <<"Number of channels:"<< channels;
5. **for** (intc = 0; c < channels; ++c) {
6. **for** (inti= 0; i< dim; ++i) {
7. mean\_values[c] += sum\_blob.data(dim \* c + i);
8. }
9. LOG(INFO) <<"mean\_value channel["<< c <<"]:"<<mean\_values[c]/ dim;
10. }

三，相关文件

compute\_image\_mean.cpp

**[cpp]** [view plaincopy](http://blog.csdn.net/whiteinblue/article/details/45540301)[在CODE上查看代码片](https://code.csdn.net/snippets/660531)

1. #include <stdint.h>
2. #include <algorithm>
3. #include <string>
4. #include <utility>
5. #include <vector>
7. #include "boost/scoped\_ptr.hpp"
8. #include "gflags/gflags.h"
9. #include "glog/logging.h"
11. #include "caffe/proto/caffe.pb.h"
12. #include "caffe/util/db.hpp"
13. #include "caffe/util/io.hpp"
15. **using** **namespace** caffe;  // NOLINT(build/namespaces)
17. **using** std::max;
18. **using** std::pair;
19. **using** boost::scoped\_ptr;
21. DEFINE\_string(backend, "lmdb",
22. "The backend {leveldb, lmdb} containing the images");
24. **int** main(**int** argc, **char**\*\* argv) {
25. ::google::InitGoogleLogging(argv[0]);
27. #ifndef GFLAGS\_GFLAGS\_H\_
28. **namespace** gflags = google;
29. #endif
31. gflags::SetUsageMessage("Compute the mean\_image of a set of images given by"
32. " a leveldb/lmdb\n"
33. "Usage:\n"
34. "    compute\_image\_mean [FLAGS] INPUT\_DB [OUTPUT\_FILE]\n");
36. gflags::ParseCommandLineFlags(&argc, &argv, **true**);
38. **if** (argc < 2 || argc > 3) {
39. gflags::ShowUsageWithFlagsRestrict(argv[0], "tools/compute\_image\_mean");
40. **return** 1;
41. }
43. scoped\_ptr<db::DB> db(db::GetDB(FLAGS\_backend));
44. db->Open(argv[1], db::READ);
45. scoped\_ptr<db::Cursor> cursor(db->NewCursor());
47. BlobProto sum\_blob;
48. **int** count = 0;
49. // load first datum
50. Datum datum;
51. datum.ParseFromString(cursor->value());
53. **if** (DecodeDatumNative(&datum)) {
54. LOG(INFO) << "Decoding Datum";
55. }
57. sum\_blob.set\_num(1);
58. sum\_blob.set\_channels(datum.channels());
59. sum\_blob.set\_height(datum.height());
60. sum\_blob.set\_width(datum.width());
61. **const** **int** data\_size = datum.channels() \* datum.height() \* datum.width();
62. **int** size\_in\_datum = std::max<**int**>(datum.data().size(),datum.float\_data\_size());
63. **for** (**int** i = 0; i < size\_in\_datum; ++i) {
64. sum\_blob.add\_data(0.);//设置初值为float型的0.0
65. }
66. LOG(INFO) << "Starting Iteration";
67. **while** (cursor->valid()) {//如果cursor是有效的
68. Datum datum;
69. datum.ParseFromString(cursor->value());//解析cuisor.value返回的字符串值，到datum
70. DecodeDatumNative(&datum);
72. **const** std::string& data = datum.data();//利用data来引用datum.data
73. size\_in\_datum = std::max<**int**>(datum.data().size(),datum.float\_data\_size());
74. CHECK\_EQ(size\_in\_datum, data\_size) << "Incorrect data field size " <<size\_in\_datum;
75. **if** (data.size() != 0) {
76. CHECK\_EQ(data.size(), size\_in\_datum);
77. **for** (**int** i = 0; i < size\_in\_datum; ++i) {
78. sum\_blob.set\_data(i, sum\_blob.data(i) + (uint8\_t)data[i]);
79. }
80. } **else** {
81. CHECK\_EQ(datum.float\_data\_size(), size\_in\_datum);
82. **for** (**int** i = 0; i < size\_in\_datum; ++i) {
83. sum\_blob.set\_data(i, sum\_blob.data(i) +
84. **static\_cast**<**float**>(datum.float\_data(i)));
85. }
86. }
87. ++count;
88. **if** (count % 10000 == 0) {
89. LOG(INFO) << "Processed " << count << " files.";
90. }
91. cursor->Next();
92. }
94. **if** (count % 10000 != 0) {
95. LOG(INFO) << "Processed " << count << " files.";
96. }
97. **for** (**int** i = 0; i < sum\_blob.data\_size(); ++i) {
98. sum\_blob.set\_data(i, sum\_blob.data(i) / count);
99. }
100. // Write to disk
101. **if** (argc == 3) {
102. LOG(INFO) << "Write to " << argv[2];
103. WriteProtoToBinaryFile(sum\_blob, argv[2]);
104. }
105. **const** **int** channels = sum\_blob.channels();
106. **const** **int** dim = sum\_blob.height() \* sum\_blob.width();
107. std::vector<**float**> mean\_values(channels, 0.0);
108. LOG(INFO) << "Number of channels: " << channels;
109. **for** (**int** c = 0; c < channels; ++c) {
110. **for** (**int** i = 0; i < dim; ++i) {
111. mean\_values[c] += sum\_blob.data(dim \* c + i);
112. }
113. LOG(INFO) << "mean\_value channel [" << c << "]:" << mean\_values[c] / dim;
114. }
115. **return** 0;
116. }

四：以上代码注释为个人理解，如有遗漏，错误还望大家多多交流，指正，以便共同学习，进步！！