Caffe2——cifar10数据集创建lmdb或leveldb类型的数据

标签：class log com 代码 使用 src http si html

Caffe2——cifar10数据集创建lmdb或leveldb类型的数据

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

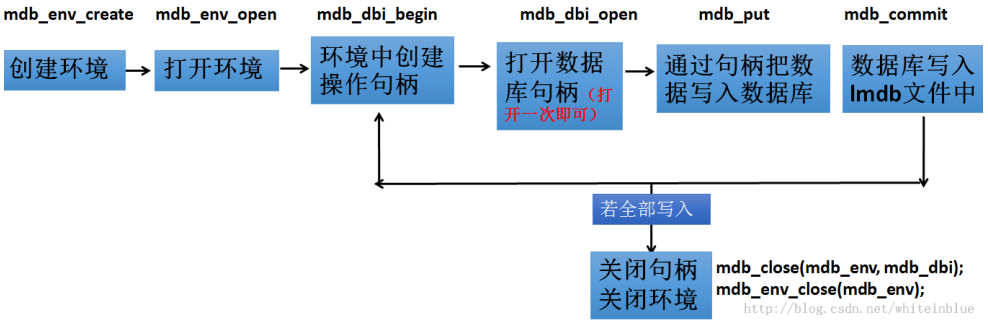
和转换mnist数据不同的是，cifar并没有使用gflags命令行解析工具；所以也没有通过gflags的宏定义来指定要转换的数据类型，而是把转换的类型参数直接作为main（）函数的参数（这种方式便于理解）。

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

```
#include <fstream>
#include <string>
#include "boost/scoped_ptr.hpp"
#include "glog/logging.h"
#include "google/protobuf/text_format.h"
#include "stdint.h"
#include "caffe/proto/caffe.pb.h"
#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

```
using caffe::Datum;
using boost::scoped_ptr;
using std::string;
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

```
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

```
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

```
void LMDB::Open(const string& source, Mode mode) {
  MDB_CHECK(mdb_env_create(&mdb_env_));//创建lmdb操作环境
 MDB_CHECK(mdb_env_set_mapsize(mdb_env_, LMDB_MAP_SIZE));//设置环境内训映射
  if (mode == NEW) {
    CHECK_EQ(mkdir(source.c_str(),0744), 0) << "mkdir " << source <<"failed";
  }//检查文件
  int flags = 0;
  if (mode == READ) {
    flags = MDB_RDONLY | MDB_NOTLS;
  }
  MDB_CHECK(mdb_env_open(mdb_env_,source.c_str(), flags, 0664));//打开创建的环境
  LOG(INFO) << "Openedlmdb " << source;
}
```

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

```
LMDBTransaction* LMDB::NewTransaction() {
  MDB_txn* mdb_txn;
  MDB_CHECK(mdb_txn_begin(mdb_env_,NULL, 0, &mdb_txn));//创建操作句柄
  MDB_CHECK(mdb_dbi_open(mdb_txn,NULL, 0, &mdb_dbi_));//打开数据库环境
  return new LMDBTransaction(&mdb_dbi_,mdb_txn);
```

}

3.3 定义数据结构文件

[cpp] view plaincopy

```
const int kCIFARSize =32;
const intkCIFARImageNBytes = 3072; //32*32=1024，RGB各占一个字节,感觉应该为uint8_t，0~255，
const intkCIFARBatchSize = 10000; //cifar共计5万个训练样本，分成5份batches，每份1万个
const int kCIFARTrainBatches= 5;
 // Data buffer
 int label;
 charstr_buffer[kCIFARImageNBytes]; //定义字符数组，一个数组可以存放一张图片的数据
 Datum datum;
 datum.set_channels(3);
 datum.set_height(kCIFARSize);
 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

```
for (int fileid = 0;fileid < kCIFARTrainBatches; ++fileid) {
snprintf(str_buffer, kCIFARImageNBytes, "/data_batch_%d.bin", fileid + 1);
std::ifstream data_file((input_folder + str_buffer).c_str(),std::ios::in| std::ios::binary);
   CHECK(data_file) << "Unable to open train file #" <<fileid + 1;
       //str_buffer=/data_batch_1.bin,等等，但str_buffer是个字符数组
       //以二进制和流输入的方式打开文件data/cifar10/data_batch_1.bin
       //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

```
for (int itemid = 0;itemid < kCIFARBatchSize; ++itemid) {
    read_image(&data_file, &label,str_buffer);
//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
void read_image(std::ifstream* file,int* label, char*buffer) {
        charlabel_char;
        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
        *label = label_char; //把label_char的值，给label
        file->read(buffer,kCIFARImageNBytes);
        return;
        }
```

3.6 读取的数据赋值到“转换”数据对象datum，并序列化

[cpp] view plaincopy

```
datum.set_label(label);
datum.set_data(str_buffer,kCIFARImageNBytes);
string out;
CHECK(datum.SerializeToString(&out));
```

3.7 把数据写入数据库

[cpp] view plaincopy

```
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

```
string out;
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

```
void LMDBTransaction::Put(conststring& key,const string& value) {
  MDB_val mdb_key, mdb_value;//声明MDB_val不透明类型数据结构“对象”
  mdb_key.mv_data = const_cast<char*>(key.data());//通过指针的方式给mdb_key赋值
  mdb_key.mv_size = key.size();
  mdb_value.mv_data = const_cast<char*>(value.data());
  mdb_value.mv_size = value.size();
  MDB_CHECK(mdb_put(mdb_txn_, *mdb_dbi_,&mdb_key, &mdb_value, 0));
//通过mdb_put（ ）句柄把mdb_key和mdb_value中的数据，写入数据库中
}
```

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

```
txn->Commit();
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

```
// This script converts the CIFAR dataset to the leveldb format used
// by caffe to perform classification.
// Usage:
//    convert_cifar_data input_folder output_db_file
// The CIFAR dataset could be downloaded at
//    http://www.cs.toronto.edu/~kriz/cifar.html

#include <fstream>  // NOLINT(readability/streams)，文件输入输出必备的文件流
#include <string>

#include "boost/scoped_ptr.hpp"//智能指针
#include "glog/logging.h"//用于日志记录，具体记录什么不是很清楚，
```

```
#include "google/protobuf/text_format.h"//用于解析.prototxt文件的
#include "stdint.h"

#include "caffe/proto/caffe.pb.h" //解析.prototxt文件的头文件
#include "caffe/util/db.hpp" //db.cpp文件中定义了NewTransaction（ ），Open（ ）等leveldb和lmdb操
作函数

using caffe::Datum;
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。
using std::string;
namespace db = caffe::db;//引入caffe命名空间中的db子命名空间

const int kCIFARSize = 32;
const int kCIFARImageNBytes = 3072;//32*32=1024，RGB各占一个字节,感觉应该为uint8_t，0~255，
const int kCIFARBatchSize = 10000;//cifar共计5万个训练样本，分成5份batches，每份1万个，
const int kCIFARTrainBatches = 5;


 void read_image(std::ifstream* file, int* label, char* buffer) {
    char label_char;
    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
    *label = label_char;//把label_char的值，给label
    file->read(buffer, kCIFARImageNBytes);
    return;
    }

//以值引用的方式传递参数（string& input_folder），
void convert_dataset(const string& input_folder, const string& output_folder,
    const string& db_type) {
  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。
  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
  scoped_ptr<db::Transaction> txn(train_db->NewTransaction());//这个transaction暂时不清楚是干什么
用的
  // Data buffer
  int label;
  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
  Datum datum;
  datum.set_channels(3);
  datum.set_height(kCIFARSize);
  datum.set_width(kCIFARSize);

  LOG(INFO) << "Writing Training data";
  for (int fileid = 0; fileid < kCIFARTrainBatches; ++fileid) {//依次遍历每个batches，共计5个
    // Open files
    LOG(INFO) << "Training Batch " << fileid + 1;
```

```
    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
    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
        std::ios::in | std::ios::binary);//c_str() 以 char* 形式传回 string 内含字符串
    CHECK(data_file) << "Unable to open train file #" << fileid + 1;
    for (int itemid = 0; itemid < kCIFARBatchSize; ++itemid) {
      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
      datum.set_label(label);
      datum.set_data(str_buffer, kCIFARImageNBytes);
      int length = snprintf(str_buffer, kCIFARImageNBytes, "%05d",
          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
      string out;
      CHECK(datum.SerializeToString(&out));
      txn->Put(string(str_buffer, length), out);//string(str_buffer, length)用来截取str_buffer的前length个
字符；
    }
  }
  txn->Commit();
  train_db->Close();

  LOG(INFO) << "Writing Testing data";
  scoped_ptr<db::DB> test_db(db::GetDB(db_type));
  test_db->Open(output_folder + "/cifar10_test_" + db_type, db::NEW);
  txn.reset(test_db->NewTransaction());
  // Open files
  std::ifstream data_file((input_folder + "/test_batch.bin").c_str(),
      std::ios::in | std::ios::binary);
  CHECK(data_file) << "Unable to open test file.";
  for (int itemid = 0; itemid < kCIFARBatchSize; ++itemid) {
    read_image(&data_file, &label, str_buffer);
    datum.set_label(label);
    datum.set_data(str_buffer, kCIFARImageNBytes);
    int length = snprintf(str_buffer, kCIFARImageNBytes, "%05d", itemid);
    string out;
    CHECK(datum.SerializeToString(&out));
    txn->Put(string(str_buffer, length), out);
  }
  txn->Commit();
  test_db->Close();
}

int main(int argc, char** argv) {
  if (argc != 4) {
    printf("This script converts the CIFAR dataset to the leveldb format used\n"
           "by caffe to perform classification.\n"
```

```
        "Usage:\n"
        "    convert_cifar_data input_folder output_folder db_type\n"
        "Where the input folder should contain the binary batch files.\n"
        "The CIFAR dataset could be downloaded at\n"
        "    http://www.cs.toronto.edu/~kriz/cifar.html\n"
        "You should gunzip them after downloading.\n");
  } else {
    google::InitGoogleLogging(argv[0]);
    convert_dataset(string(argv[1]), string(argv[2]), string(argv[3]));
    //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]；
    //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。
  }
  return 0;
}
```

db.cpp 文件

里面定义了caffe名字空间和其子空间db

[cpp] view plaincopy

```
#include "caffe/util/db.hpp"

#include <sys/stat.h>
#include <string>

namespace caffe { namespace db {

const size_t LMDB_MAP_SIZE = 1099511627776;  // 1 TB

//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
void LevelDB::Open(const string& source, Mode mode) {
  leveldb::Options options;//创建leveldb中的options类型对象
  options.block_size = 65536;
  options.write_buffer_size = 268435456;
  options.max_open_files = 100;
  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
  options.create_if_missing = mode != READ;//
  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
  CHECK(status.ok()) << "Failed to open leveldb " << source
                     << std::endl << status.ToString();
```

```
  LOG(INFO) << "Opened leveldb " << source;
}


//Open函数主要负责，创建环境；设置环境参数，打开环境
void LMDB::Open(const string& source, Mode mode) {
  MDB_CHECK(mdb_env_create(&mdb_env_));//创建lmdb操作环境
  MDB_CHECK(mdb_env_set_mapsize(mdb_env_, LMDB_MAP_SIZE));//设置环境内训映射
  if (mode == NEW) {
    CHECK_EQ(mkdir(source.c_str(), 0744), 0) << "mkdir " << source << "failed";
  }//检查文件
  int flags = 0;
  if (mode == READ) {
    flags = MDB_RDONLY | MDB_NOTLS;
  }
  MDB_CHECK(mdb_env_open(mdb_env_, source.c_str(), flags, 0664));//打开创建的环境
  LOG(INFO) << "Opened lmdb " << source;
}

LMDBCursor* LMDB::NewCursor() {
  MDB_txn* mdb_txn;
  MDB_cursor* mdb_cursor;
  MDB_CHECK(mdb_txn_begin(mdb_env_, NULL, MDB_RDONLY, &mdb_txn));
  MDB_CHECK(mdb_dbi_open(mdb_txn, NULL, 0, &mdb_dbi_));
  MDB_CHECK(mdb_cursor_open(mdb_txn, mdb_dbi_, &mdb_cursor));
  return new LMDBCursor(mdb_txn, mdb_cursor);
}

//在lmdb环境中创建操作句柄
LMDBTransaction* LMDB::NewTransaction() {
  MDB_txn* mdb_txn;
  MDB_CHECK(mdb_txn_begin(mdb_env_, NULL, 0, &mdb_txn));//创建操作句柄
  MDB_CHECK(mdb_dbi_open(mdb_txn, NULL, 0, &mdb_dbi_));//打开数据库环境
  return new LMDBTransaction(&mdb_dbi_, mdb_txn);
}

void LMDBTransaction::Put(const string& key, const string& value) {
  MDB_val mdb_key, mdb_value;
  mdb_key.mv_data = const_cast<char*>(key.data());
  mdb_key.mv_size = key.size();
  mdb_value.mv_data = const_cast<char*>(value.data());
  mdb_value.mv_size = value.size();
  MDB_CHECK(mdb_put(mdb_txn_, *mdb_dbi_, &mdb_key, &mdb_value, 0));
}

DB* GetDB(DataParameter::DB backend) {
  switch (backend) {
  case DataParameter_DB_LEVELDB:
```

```
    return new LevelDB();
  case DataParameter_DB_LMDB:
    return new LMDB();
  default:
    LOG(FATAL) << "Unknown database backend";
  }
}

//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
DB* GetDB(const string& backend) {
  if (backend == "leveldb") {
    return new LevelDB();
  } else if (backend == "lmdb") {
    return new LMDB();
  } else {
    LOG(FATAL) << "Unknown database backend";
  }
}

}  // namespace db
}  // 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！！

Caffe2——cifar10数据集创建lmdb或leveldb类型的数据

标签 : class log com 代码 使用 src http si html