Caffe3——ImageNet数据集创建lmdb类型的数据

标签：算法 class style log com 代码 使用 src http

Caffe3——ImageNet数据集创建lmdb类型的数据

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：

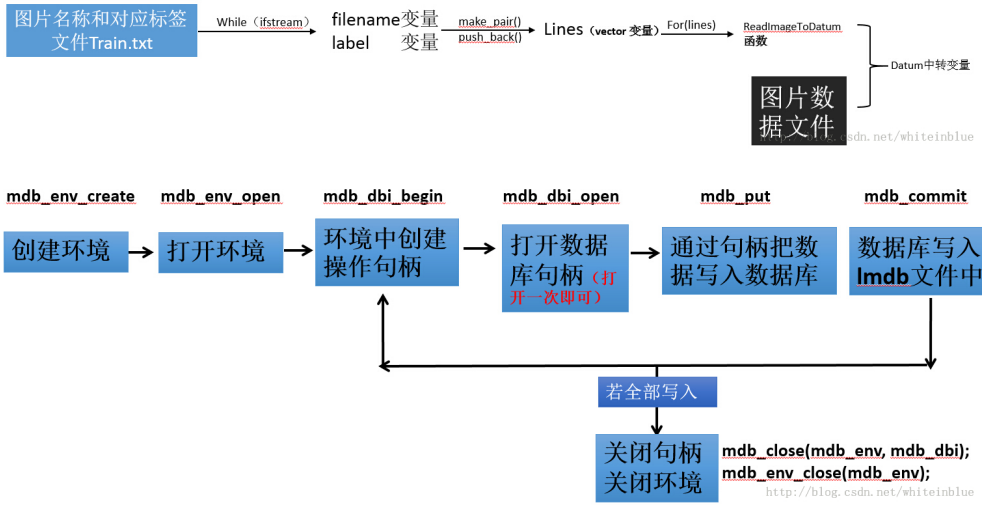
[cpp] view plaincopy

```
<pre name="code" class="cpp">GLOG_logtostderr=1$TOOLS/convert_imageset \
    --resize_height=$RESIZE_HEIGHT \
    --resize_width=$RESIZE_WIDTH \
    --shuffle \
    $TRAIN_DATA_ROOT \  图像数据集存放的根目录
$DATA/train.txt \       图像的ID和对应的分类标签数字
$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

```
#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，
#include <fstream>  // NOLINT(readability/streams)
#include <string>
#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
#include<vector>//会自动扩展容量的数组
#include "boost/scoped_ptr.hpp"//智能指针头文件
#include "gflags/gflags.h"
#include "glog/logging.h"
#include"caffe/proto/caffe.pb.h"
#include "caffe/util/db.hpp" //引入包装好的lmdb操作函数
#include "caffe/util/io.hpp" //引入opencv中的图像操作函数
#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

```
using namespace caffe;  // NOLINT(build/namespaces)
using std::pair;
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

DEFINE_bool(gray, false,"When thisoption is on, treat images as grayscale ones");//是否为灰度图片
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
DEFINE_string(backend, "lmdb","The backend {lmdb, leveldb} for storing the result");//默认转换的数据类型
DEFINE_int32(resize_width, 0, "Width images areresized to");//定义resize的尺寸，默认为0，不转换尺寸
DEFINE_int32(resize_height, 0, "Height imagesare resized to");
DEFINE_bool(check_size, false,"When this optionis on, check that all the datum have the samesize");
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
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

const boolis_color = !FLAGS_gray; //通过gflags把宏定义变量的值，赋值给常值变量
const boolcheck_size = FLAGS_check_size; //检查图像的size
const boolencoded = FLAGS_encoded;//是否编译（转换）图像格式
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

while (infile>> filename >> label) {
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

if(FLAGS_shuffle) {
// randomlyshuffle data
LOG(INFO)<< "Shuffling data";

[cpp] view plaincopy

//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

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

scoped_ptr<db::DB>db(db::GetDB(FLAGS_backend));
//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
db->Open(argv[3], db::NEW);//argv[3]的文件夹下创建并打开lmdb的操作环境

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

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

```
if (check_size) {//检查图片尺寸
    if (!data_size_initialized) {//若data_size_initialized没有初始化
      data_size = datum.channels() *datum.height() * datum.width();
      data_size_initialized = true;
    } else {
      const std::string& data =datum.data();
      CHECK_EQ(data.size(), data_size)<< "Incorrect data field size "<< data.size();
    }
```

3.6 序列化键和值并放入临时数据库

[cpp] view plaincopy

```
// sequential
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
   // Put in db
```

```
 string out;
   CHECK(datum.SerializeToString(&out));//datum数据，序列化到字符串中
   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

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

四，相关文件

4.1 Convert_imageset.cpp文件

[cpp] view plaincopy

```
// This program converts a set of images to a lmdb/leveldb by storing them
// as Datum proto buffers.
// Usage:
//   convert_imageset [FLAGS] ROOTFOLDER/ LISTFILE DB_NAME
//
// where ROOTFOLDER is the root folder that holds all the images, and LISTFILE
// should be a list of files as well as their labels, in the format as
//   subfolder1/file1.JPEG 7
//   ....

#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，
#include <fstream>  // NOLINT(readability/streams)
#include <string>
#include <utility>//utility头文件定义了一个pair类型
#include <vector>//会自动扩展容量的数组

#include "boost/scoped_ptr.hpp"
#include "gflags/gflags.h"
#include "glog/logging.h"

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

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

using namespace caffe;  // NOLINT(build/namespaces)
using std::pair;
using boost::scoped_ptr;


//通过gflags宏定义一些程序的参数变量
DEFINE_bool(gray, false,
    "When this option is on, treat images as grayscale ones");
DEFINE_bool(shuffle, false,
    "Randomly shuffle the order of images and their labels");//洗牌，随机打乱数据集的顺序
DEFINE_string(backend, "lmdb",
        "The backend {lmdb, leveldb} for storing the result");
DEFINE_int32(resize_width, 0, "Width images are resized to");
DEFINE_int32(resize_height, 0, "Height images are resized to");
DEFINE_bool(check_size, false,
    "When this option is on, check that all the datum have the same size");
DEFINE_bool(encoded, false,
    "When this option is on, the encoded image will be save in datum");//用于转换数据格式的
DEFINE_string(encode_type, "",
    "Optional: What type should we encode the image as ('png','jpg',...).");//要转换的数据格式

int main(int argc, char** argv) {
  ::google::InitGoogleLogging(argv[0]);

#ifndef GFLAGS_GFLAGS_H_
  namespace gflags = google;
#endif

  gflags::SetUsageMessage("Convert a set of images to the leveldb/lmdb\n"
        "format used as input for Caffe.\n"
        "Usage:\n"
        "    convert_imageset [FLAGS] ROOTFOLDER/ LISTFILE DB_NAME\n"
        "The ImageNet dataset for the training demo is at\n"
        "    http://www.image-net.org/download-images\n");
  gflags::ParseCommandLineFlags(&argc, &argv, true);

  if (argc < 4) {
    gflags::ShowUsageWithFlagsRestrict(argv[0], "tools/convert_imageset");
    return 1;
  }
  //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
  const bool is_color = !FLAGS_gray;  //通过gflags把宏定义变量的值，赋值给常值变量
  const bool check_size = FLAGS_check_size; //检查图像的size
```

```
  const bool encoded = FLAGS_encoded;//是否编译（转换）图像格式
  const string encode_type = FLAGS_encode_type;//要编译的图像格式

  std::ifstream infile(argv[2]);//定义指向train.txt数据文件的文件读入流
  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
  std::string filename;
  int label;
  //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
  while (infile >> filename >> label) {
    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
  }
  if (FLAGS_shuffle) {
    // randomly shuffle data
    LOG(INFO) << "Shuffling data";
    //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
    shuffle(lines.begin(), lines.end());//vector.begin() - 回传一个Iterator迭代器，它指向 vector 第一个元素。
  }
  LOG(INFO) << "A total of " << lines.size() << " images.";

  if (encode_type.size() && !encoded)
    LOG(INFO) << "encode_type specified, assuming encoded=true.";

  int resize_height = std::max<int>(0, FLAGS_resize_height);
  int resize_width = std::max<int>(0, FLAGS_resize_width);

  // Create new DB
  scoped_ptr<db::DB> db(db::GetDB(FLAGS_backend));
  db->Open(argv[3], db::NEW);//argv[3]的文件夹下打开创建lmdb的操作环境
  scoped_ptr<db::Transaction> txn(db->NewTransaction());//创建lmdb文件的操作句柄

  // Storing to db
  std::string root_folder(argv[1]);//把源数据文件的地址复制给root_folder
  Datum datum;//声明数据“转换”对象
  int count = 0;
  const int kMaxKeyLength = 256;
  char key_cstr[kMaxKeyLength];
  int data_size = 0;
  bool data_size_initialized = false;

  for (int line_id = 0; line_id < lines.size(); ++line_id) {
    bool status;
    std::string enc = encode_type; //enc为空串，则enc.size()=false;否则为true
    if (encoded && !enc.size()) {
      // Guess the encoding type from the file name
```

```
    string fn = lines[line_id].first;//把图像的文件名赋值给fn（filename）
    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
    if ( p == fn.npos )
      LOG(WARNING) << "Failed to guess the encoding of '" << fn << "'";
    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
    std::transform(enc.begin(), enc.end(), enc.begin(), ::tolower);//将enc字符串转换成小写
   }
   //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）
   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
   if (status == false) continue;//status=false,说明此张图片读取错误；“跳过”继续下一张
   if (check_size) {//检查图片尺寸
    if (!data_size_initialized) {//若data_size_initialized没有初始化
     data_size = datum.channels() * datum.height() * datum.width();
     data_size_initialized = true;
    } else {
     const std::string& data = datum.data();
     CHECK_EQ(data.size(), data_size) << "Incorrect data field size "
        << data.size();
    }
   }
   // sequential
   int length = 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

   // Put in db
   string out;
   CHECK(datum.SerializeToString(&out));//datum数据，序列化到字符串中
   txn->Put(string(key_cstr, length), out);//把键值对放入到数据库

   if (++count % 1000 == 0) {
    // Commit db
    txn->Commit();//保存到lmdb类型的文件
    txn.reset(db->NewTransaction());//重新初始化操作句柄
    LOG(ERROR) << "Processed " << count << " files.";
   }
  }
  // write the last batch
  if (count % 1000 != 0) {
   txn->Commit();
   LOG(ERROR) << "Processed " << count << " files.";
  }
  return 0;
}
```

4.2 io.cpp文件

[cpp] view plaincopy

```
#include <fcntl.h>
#include <google/protobuf/io/coded_stream.h>
#include <google/protobuf/io/zero_copy_stream_impl.h>
#include <google/protobuf/text_format.h>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/highgui/highgui_c.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <stdint.h>

#include <algorithm>
#include <fstream>  // NOLINT(readability/streams)
#include <string>
#include <vector>

#include "caffe/common.hpp"
#include "caffe/proto/caffe.pb.h"
#include "caffe/util/io.hpp"

const int kProtoReadBytesLimit = INT_MAX;  // Max size of 2 GB minus 1 byte.

namespace caffe {

using google::protobuf::io::FileInputStream;//文件输入流
using google::protobuf::io::FileOutputStream;//文件输出流
using google::protobuf::io::ZeroCopyInputStream;//These interfaces are different from classic I/O stre
ams in that they try to minimize the amount of data copying that needs to be done
using google::protobuf::io::CodedInputStream;
using google::protobuf::io::ZeroCopyOutputStream;
using google::protobuf::io::CodedOutputStream;
using google::protobuf::Message;

bool ReadProtoFromTextFile(const char* filename, Message* proto) {//从文本文件中读入proto文件
  int fd = open(filename, O_RDONLY);
  CHECK_NE(fd, -1) << "File not found: " << filename;
  FileInputStream* input = new FileInputStream(fd);
  bool success = google::protobuf::TextFormat::Parse(input, proto);
  delete input;
  close(fd);
  return success;
}
```

```
void WriteProtoToTextFile(const Message& proto, const char* filename) {//想文本文件中写入proto文
件
  int fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 0644);
  FileOutputStream* output = new FileOutputStream(fd);
  CHECK(google::protobuf::TextFormat::Print(proto, output));
  delete output;
  close(fd);
}

bool ReadProtoFromBinaryFile(const char* filename, Message* proto) {//从二进制文件读入proto
  int fd = open(filename, O_RDONLY);
  CHECK_NE(fd, -1) << "File not found: " << filename;
  ZeroCopyInputStream* raw_input = new FileInputStream(fd);
  CodedInputStream* coded_input = new CodedInputStream(raw_input);
  coded_input->SetTotalBytesLimit(kProtoReadBytesLimit, 536870912);

  bool success = proto->ParseFromCodedStream(coded_input);

  delete coded_input;
  delete raw_input;
  close(fd);
  return success;
}

void WriteProtoToBinaryFile(const Message& proto, const char* filename) {//把proto写入二进制文件
中
  fstream output(filename, ios::out | ios::trunc | ios::binary);
  CHECK(proto.SerializeToOstream(&output));
}


//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
//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）。
//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，
//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，
//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。
cv::Mat ReadImageToCVMat(const string& filename,
    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
  cv::Mat cv_img;
  int cv_read_flag = (is_color ? CV_LOAD_IMAGE_COLOR :
    CV_LOAD_IMAGE_GRAYSCALE);
  cv::Mat cv_img_origin = cv::imread(filename, cv_read_flag);//读取图片内容
  if (!cv_img_origin.data) {
```

```
    LOG(ERROR) << "Could not open or find file " << filename;
    return cv_img_origin;
  }
  if (height > 0 && width > 0) {
    cv::resize(cv_img_origin, cv_img, cv::Size(width, height));
  } else {
    cv_img = cv_img_origin;
  }
  return cv_img;
}

cv::Mat ReadImageToCVMat(const string& filename,//读取图片到CVMat中，重载1
    const int height, const int width) {
  return ReadImageToCVMat(filename, height, width, true);
}

cv::Mat ReadImageToCVMat(const string& filename,//读取图片到CVMat中，重载2
    const bool is_color) {
  return ReadImageToCVMat(filename, 0, 0, is_color);
}

cv::Mat ReadImageToCVMat(const string& filename) {//读取图片到CVMat中，重载3
  return ReadImageToCVMat(filename, 0, 0, true);
}
// Do the file extension and encoding match?
static bool matchExt(const std::string & fn, //匹配拓展名称？
                     std::string en) {
  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
  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
  std::transform(ext.begin(), ext.end(), ext.begin(), ::tolower);//把ext变成小写
  std::transform(en.begin(), en.end(), en.begin(), ::tolower);
  if ( ext == en )
    return true;
  if ( en == "jpg" && ext == "jpeg" )
    return true;
  return false;
}

bool ReadImageToDatum(const string& filename, const int label,//把图片读到 Datum中
    const int height, const int width, const bool is_color,
    const std::string & encoding, Datum* datum) {
  cv::Mat cv_img = ReadImageToCVMat(filename, height, width, is_color);//先把数据读到cv::Mat类型矩
阵中
  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
    if (encoding.size()) {//是否需要编码
      if ( (cv_img.channels() == 3) == is_color && !height && !width &&
          matchExt(filename, encoding) )
```

```
      return ReadFileToDatum(filename, label, datum);

    std::vector<uchar> buf;
    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
    datum->set_data(std::string(reinterpret_cast<char*>(&buf[0]),
                buf.size()));
    datum->set_label(label);
    datum->set_encoded(true);//感觉是一种编码函数
    return true;
   }
   CVMatToDatum(cv_img, datum);
   datum->set_label(label);
   return true;
  } else {
   return false;
  }
}

bool ReadFileToDatum(const string& filename, const int label,
   Datum* datum) {
  std::streampos size;

  fstream file(filename.c_str(), ios::in|ios::binary|ios::ate);
  if (file.is_open()) {
   size = file.tellg();
   std::string buffer(size, ' ');
   file.seekg(0, ios::beg);
   file.read(&buffer[0], size);
   file.close();
   datum->set_data(buffer);
   datum->set_label(label);
   datum->set_encoded(true);
   return true;
  } else {
   return false;
  }
}

cv::Mat DecodeDatumToCVMatNative(const Datum& datum) {
  cv::Mat cv_img;
  CHECK(datum.encoded()) << "Datum not encoded";
  const string& data = datum.data();
  std::vector<char> vec_data(data.c_str(), data.c_str() + data.size());
  cv_img = cv::imdecode(vec_data, -1);
  if (!cv_img.data) {
   LOG(ERROR) << "Could not decode datum ";
  }
  return cv_img;
```

```
}
cv::Mat DecodeDatumToCVMat(const Datum& datum, bool is_color) {
  cv::Mat cv_img;
  CHECK(datum.encoded()) << "Datum not encoded";
  const string& data = datum.data();
  std::vector<char> vec_data(data.c_str(), data.c_str() + data.size());
  int cv_read_flag = (is_color ? CV_LOAD_IMAGE_COLOR :
    CV_LOAD_IMAGE_GRAYSCALE);
  cv_img = cv::imdecode(vec_data, cv_read_flag);
  if (!cv_img.data) {
    LOG(ERROR) << "Could not decode datum ";
  }
  return cv_img;
}

// If Datum is encoded will decoded using DecodeDatumToCVMat and CVMatToDatum
// If Datum is not encoded will do nothing
bool DecodeDatumNative(Datum* datum) {
  if (datum->encoded()) {
    cv::Mat cv_img = DecodeDatumToCVMatNative((*datum));
    CVMatToDatum(cv_img, datum);
    return true;
  } else {
    return false;
  }
}
bool DecodeDatum(Datum* datum, bool is_color) {
  if (datum->encoded()) {
    cv::Mat cv_img = DecodeDatumToCVMat((*datum), is_color);
    CVMatToDatum(cv_img, datum);
    return true;
  } else {
    return false;
  }
}

void CVMatToDatum(const cv::Mat& cv_img, Datum* datum) {
  CHECK(cv_img.depth() == CV_8U) << "Image data type must be unsigned byte";
  datum->set_channels(cv_img.channels());
  datum->set_height(cv_img.rows);
  datum->set_width(cv_img.cols);
  datum->clear_data();
  datum->clear_float_data();
  datum->set_encoded(false);
  int datum_channels = datum->channels();
  int datum_height = datum->height();
  int datum_width = datum->width();
  int datum_size = datum_channels * datum_height * datum_width;
```

```
  std::string buffer(datum_size, ' ');
  for (int h = 0; h < datum_height; ++h) {
    const uchar* ptr = cv_img.ptr<uchar>(h);
    int img_index = 0;
    for (int w = 0; w < datum_width; ++w) {
      for (int c = 0; c < datum_channels; ++c) {
        int datum_index = (c * datum_height + h) * datum_width + w;
        buffer[datum_index] = static_cast<char>(ptr[img_index++]);
      }
    }
  }
  datum->set_data(buffer);
}
```

。。。。。

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

标签：算法 class style log com 代码 使用 src http