xiaoYY

< 2015年8月 >
日 一 二 三 四 五 六
26 27 28 29 30 31 1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31 1 2 3 4 5

昵称：xiaoYY
园龄：2年10个月
粉丝：12
关注：24
+加关注

搜索
找找看
谷歌搜索

常用链接
我的随笔
我的评论
我的参与
最新评论
我的标签
更多链接

随笔分类
C++(7)
caffe（Linux）下操作（转载）(7)
caffe（Windows）下操作（转载）(1)
caffe（转载）(5)
caffe(自带模型解读)（转载）(6)
caffe配套基本操作(12)
caffe源码解析（转载）(5)
CNN(6)
debug(1)
Java(1)
Linux(13)
Matlab(46)
NLP(5)
Office(5)
PRML(3)
Protocol Buffer(7)
PS(2)
Python(17)
RNN(7)
SVM(11)
UFLDL
UFLDL（转载）(2)
报告(2)
编程技巧经验(13)
并行计算(1)
程序设计小问题(3)
吃(3)
处理数据(23)
大牛对话(5)
大牛列表 传记 故事 (8)
代码注释
代码注释（转载）
电脑基础知识(4)
调参(35)
感悟（转载）(6)
搞笑 段子 故事(7)
各种算法（转载）(5)
会议记录(5)
机器学习(76)
基金(5)
集成学习(3)
健康养生安全(6)
降维(7)
经济(2)
卷积(8)
开会(4)
科研经验(50)
可视化(9)
励志(8)
流行学习(1)
论文代码(1)
论文目录(7)

Caffe4——计算图像均值

Caffe4——计算图像均值

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：

```
EXAMPLE=examples/imagenet
DATA=data/ilsvrc12
TOOLS=build/tools
$TOOLS/compute_image_mean $EXAMPLE/ilsvrc12_train_lmdb \
$DATA/imagenet_mean.binaryproto<strong>
</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

```
#include<stdint.h>//定义了几种扩展的整数类型和宏
#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，
#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;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。
#include<vector>//可以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"//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
usingnamespacecaffe;  //引入caffe命名空间
usingstd::max;//
usingstd::pair;
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

```
scoped_ptr<db::DB>db(db::GetDB(FLAGS_backend));
db->Open(argv[1], db::READ);//只读的方式打开lmdb文件
scoped_ptr<db::Cursor> cursor(db->NewCursor());
//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

```
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
int count = 0;
// load first datum
  Datum datum;
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

```
sum_blob.set_num(1);//设置图片的个数
sum_blob.set_channels(datum.channels());
sum_blob.set_height(datum.height());
sum_blob.set_width(datum.width());
constintdata_size = datum.channels() *datum.height() * datum.width();//每张图片的尺寸
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

```
for (inti= 0; i<size_in_datum; ++i) {
sum_blob.add_data(0.);//设置初值为float型的0.0
 }
```

2.3.6利用循环和cursor读取lmdb中的数据

```
while (cursor->valid()) {//如果cursor是有效的
    Datum datum;
datum.ParseFromString(cursor->value());//解析cuisor.value返回的字符串值，到datum
DecodeDatumNative(&datum);//感觉是把datum中字符串类型的值，变成相应的类型
conststd::string& data =datum.data();//利用data来引用datum.data
size_in_datum = std::max<int>(datum.data().size(),datum.float_data_size());
    CHECK_EQ(size_in_datum,data_size) <<"Incorrect data field size"<<size_in_datum;
if (data.size() != 0) {//datum.data().size()!=0
      CHECK_EQ(data.size(),size_in_datum);//判断是否相等
for (inti= 0; i<size_in_datum; ++i) {
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
      }
    } else{
     CHECK_EQ(datum.float_data_size(), size_in_datum);
for (inti= 0; i<size_in_datum; ++i) {
sum_blob.set_data(i, sum_blob.data(i) +
static_cast<float>(datum.float_data(i)));//对应位置的像素值相加（float类型相加）
      }
    }
    ++count;
if (count % 10000 == 0) {
LOG(INFO) <<"Processed "<<count <<" files.";
    }
    cursor->Next();//光标下移（指针），指向下一个存储在lmdb中的数据
  }
```

2.3.7 求均值

```
for (inti= 0; i<sum_blob.data_size(); ++i) {
```

```
  sum_blob.set_data(i, sum_blob.data(i) / count);
  }
```

2.3.8 存储到指定文件

```
// Write to disk
if (argc == 3) {
LOG(INFO) <<"Write to "<<argv[2];
WriteProtoToBinaryFile(sum_blob, argv[2]);
  }
```

2.3.9 计算每个channel的均值，这个貌似没有用到吧！

```
constint channels = sum_blob.channels();
constint dim = sum_blob.height() *sum_blob.width();
std::vector<float>mean_values(channels,0.0);//容量为3的数组，初始值为0.0
LOG(INFO) <<"Number of channels:"<< channels;
for (intc = 0; c < channels; ++c) {
for (inti= 0; i< dim; ++i) {
mean_values[c] += sum_blob.data(dim * c + i);
    }
LOG(INFO) <<"mean_value channel["<< c <<"]:"<<mean_values[c]/ dim;
  }
```

三，相关文件

compute_image_mean.cpp

```
#include <stdint.h>
#include <algorithm>
#include <string>
#include <utility>
#include <vector>

#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"

using namespace caffe;  // NOLINT(build/namespaces)

using std::max;
using std::pair;
using boost::scoped_ptr;

DEFINE_string(backend, "lmdb",
        "The backend {leveldb, lmdb} containing the images");

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

#ifndef GFLAGS_GFLAGS_H_
  namespace gflags = google;
#endif

  gflags::SetUsageMessage("Compute the mean_image of a set of images given by"
        " a leveldb/lmdb\n"
        "Usage:\n"
        "    compute_image_mean [FLAGS] INPUT_DB [OUTPUT_FILE]\n");

  gflags::ParseCommandLineFlags(&argc, &argv, true);

```

```
38.  if (argc < 2 || argc > 3) {
39.    gflags::ShowUsageWithFlagsRestrict(argv[0], "tools/compute_image_mean");
40.    return 1;
41.  }
42.
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());
46.
47.  BlobProto sum_blob;
48.  int count = 0;
49.  // load first datum
50.  Datum datum;
51.  datum.ParseFromString(cursor->value());
52.
53.  if (DecodeDatumNative(&datum)) {
54.    LOG(INFO) << "Decoding Datum";
55.  }
56.
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);
71.
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.  }
93.
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.   }
```

```
    LOG(INFO) << "mean_value channel [" << c << "]:" << mean_values[c] / dim;
  }
  return 0;
}
```

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！！

转载请标明出处：**http://blog.csdn.net/whiteinblue/article/details/45540301**

分类: caffe源码解析（转载）

xiaoYY
关注 - 24
粉丝 - 12
+加关注

« 上一篇：图像视频基础知识
» 下一篇：总结一下用caffe跑图片数据的研究流程

posted @ 2015-05-14 15:13 xiaoYY 阅读(112) 评论(0) 编辑 收藏