[caffe的训练之一，数据的组织。](http://blog.csdn.net/cparent/article/details/45250271)

分类： [c++](http://blog.csdn.net/xueyunf/article/category/1189786)2015-04-24 18:54 362人阅读 [评论](http://blog.csdn.net/cparent/article/details/45250271#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/cparent/article/details/45250271#report)

[caffe 深度学习](http://www.csdn.net/tag/caffe%20%e6%b7%b1%e5%ba%a6%e5%ad%a6%e4%b9%a0)[c++](http://www.csdn.net/tag/c%2b%2b)

本教程为了那些第一次使用caffe框架进行深度学习训练的人而生的，我来一个简单关于caffe训练数据的组织来个简单的介绍。我们都知道caffe中使用leveldb 和lmd两种方式进行组织数据。这里介绍一种使用lmd进行训练数据组织的方式。

我来贴一段简单的代码

组织数据。

1.     lmd的打开和写入数据

定义环境lmd的环境

MDB\_env \*mdb\_env;

 MDB\_dbi mdb\_dbi;

 MDB\_val mdb\_key, mdb\_data;

 MDB\_txn \*mdb\_txn;

 打开数据库

CHECK\_EQ(mdb\_env\_create(&mdb\_env),MDB\_SUCCESS) << "mdb\_env\_create failed";

   CHECK\_EQ(mdb\_env\_set\_mapsize(mdb\_env, 1099511627776), MDB\_SUCCESS)  // 1TB

       << "mdb\_env\_set\_mapsize failed";

   CHECK\_EQ(mdb\_env\_open(mdb\_env, argv[3], 0, 0664), MDB\_SUCCESS)

       << "mdb\_env\_open failed";

   CHECK\_EQ(mdb\_txn\_begin(mdb\_env, NULL, 0, &mdb\_txn), MDB\_SUCCESS)

       << "mdb\_txn\_begin failed";

   CHECK\_EQ(mdb\_open(mdb\_txn, NULL, 0, &mdb\_dbi), MDB\_SUCCESS)

       << "mdb\_open failed";

写入具体的数据

mdb\_data.mv\_size = value.size();//value是个图像数据值，它的size一般是datum.channels()\*datum.height()\*datum.width();

     mdb\_data.mv\_data = reinterpret\_cast<void\*>(&value[0]);

      mdb\_key.mv\_size = keystr.size();

     mdb\_key.mv\_data = reinterpret\_cast<void\*>(&keystr[0]);

     CHECK\_EQ(mdb\_put(mdb\_txn, mdb\_dbi, &mdb\_key, &mdb\_data, 0),MDB\_SUCCESS);

CHECK\_EQ这个东西是我自己定义的一个断言宏，嘿嘿。懒得贴出来了，大家可以自己编写自己的断言宏。

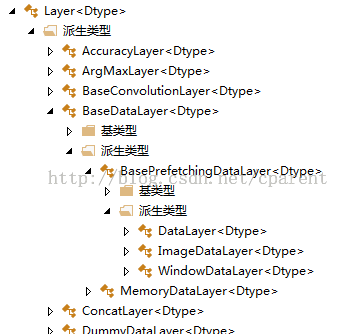
value和key均为两个string。嘿嘿。是不是觉得非常简单，引用头文件是

#include <glog/logging.h>  
#include <leveldb/db.h>  
#include <leveldb/write\_batch.h>  
#include <lmdb.h>  
#include <sys/stat.h>  
  
  
#include <algorithm>  
#include <fstream>  // NOLINT(readability/streams)  
#include <string>  
#include <utility>  
#include <vector>  
  
  
#include "caffe.pb.h"  
#include "io.hpp"  
#include "rng.hpp"

好了，到此为止大家应该知道怎么组织自己的训练数据了。提示一点，caffe的训练数据的东西都需要进行resize成同一维度的数据。

[caffe 的layer层组织结构](http://blog.csdn.net/cparent/article/details/46729423)

caffe的layer层是靠INSTANTIATE\_CLASS和REGISTER\_LAYER\_CLASS来进行对各个层类进行处理的。每一个layer都是继承基类Layer的，其中BaseDataLayer是输入数据的基类，从这个类继承的主要是：



这个图可以很好的看到这些layer的继承关系，整个layer层的结构非常清晰，使用抽象工厂的方式构建整个layer层，然后刚才的宏将layer注册成抽象的服务类，然后再使用的时候再向服务器提供者申请调用各个类，这样就可以把proto文件作为个各类的配置文件进行调用，这点和java的spring框架很像，使用的是SOA方式。

当你了解这一结构之后 ，进行对caffe layer层进行分析的时候相信大多数人会清晰很多，只不过大多数传统的c++或者java使用这种动态创建类的时候使用的是xml语言进行描述的，而caffe这个东西使用proto这种自定义的非主流的方式进行了描述。在进行整个caffe剖析的时候要将layer和其他结构独立出来进行分析。（废话：看来多学习学习设计模式等还是非常有用的。）这种设计方式的有点和缺点都很明显，可以让编译型语言在运行时有一部分灵活性，但是依然无法彻底解决灵活性问题，不过对于算法来说这个灵活性够了，也类似Strategy模式，在选择进行数据变换的时候这个数据可以有多种变换方式。而这种变换方式刚好是一种变换策略，因为网上很多人分析caffe的层，每种层的梯度如何下降，这样下降的优缺点如何太多了，我这里就不班门弄斧了，不说这些被很多人说过的东西，我只说caffe的整个工程架构。以及我们可以如何进行修改来满足我们自己的需求。

这样我们顺便看看caffe的是如何进行结构组织的，io方面使用的opencv leveldb lmdb来进行数据读取和存储。Solver这个类是最终执行所有层组合出求解过程的组合。

整个caffe的代码核心部分就在layer里面，每个layer对应一个函数的梯度下降方式，理解这些函数需要数学和传统机器学习的基础，没法短时间速成所以这里我也就不进行胡扯了。这个留作以后慢慢给大家分享，争取每周分享一个层的剖析。

这个就是整个架构的初始化各个类的宏模板函数。

**[cpp]** [view plaincopy](http://blog.csdn.net/cparent/article/details/46729423)

1. #define INSTANTIATE\_CLASS(classname) \
2. **char** gInstantiationGuard##classname; \
3. **template** **class** classname<**float**>; \
4. **template** **class** classname<**double**>

**[cpp]** [view plaincopy](http://blog.csdn.net/cparent/article/details/46729423)

1. 这个下面就是典型的SOA方式注册每个**class**，以及存放这些**class**的容器，<img src="http://img.blog.csdn.net/20150703111208565?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvY3BhcmVudA==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center" alt="" />当然这是一种简单粗暴的写法。再加上一堆模板和宏搞的整个系统看起来很高大上，其实只要对c++模板和宏熟悉，这些利器会让你的开发更轻松。

**[cpp]** [view plaincopy](http://blog.csdn.net/cparent/article/details/46729423)

1. **template** <**typename** Dtype>
2. **class** LayerRegisterer {
3. **public**:
4. LayerRegisterer(**const** string& type,
5. shared\_ptr<Layer<Dtype> > (\*creator)(**const** LayerParameter&)) {
6. // LOG(INFO) << "Registering layer type: " << type;
7. LayerRegistry<Dtype>::AddCreator(type, creator);
8. }
9. };

12. #define REGISTER\_LAYER\_CREATOR(type, creator)                                  \
13. **static** LayerRegisterer<**float**> g\_creator\_f\_##type(#type, creator<**float**>);     \
14. **static** LayerRegisterer<**double**> g\_creator\_d\_##type(#type, creator<**double**>)    \
16. #define REGISTER\_LAYER\_CLASS(type)                                             \
17. **template** <**typename** Dtype>                                                    \
18. shared\_ptr<Layer<Dtype> > Creator\_##type##Layer(**const** LayerParameter& param) \
19. {                                                                            \
20. **return** shared\_ptr<Layer<Dtype> >(**new** type##Layer<Dtype>(param));           \
21. }                                                                            \
22. REGISTER\_LAYER\_CREATOR(type, Creator\_##type##Layer)
24. }  // namespace caffe

看完这些相信大家对caffe的源码分析定位可以快速入手。

# [affe使用MemoryDataLayer从内存中加载数据](http://blog.csdn.net/cparent/article/details/47008485)

分类： [深度学习](http://blog.csdn.net/xueyunf/article/category/5623193)2015-07-22 19:53 330人阅读 [评论](http://blog.csdn.net/cparent/article/details/47008485#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/cparent/article/details/47008485#report)

[MemoryDataLayer](http://www.csdn.net/tag/MemoryDataLayer)[深度学习](http://www.csdn.net/tag/%e6%b7%b1%e5%ba%a6%e5%ad%a6%e4%b9%a0)[caffe](http://www.csdn.net/tag/caffe)[内存中加载数据](http://www.csdn.net/tag/%e5%86%85%e5%ad%98%e4%b8%ad%e5%8a%a0%e8%bd%bd%e6%95%b0%e6%8d%ae)

最近在搞caffe的应用，因为很多时候我们需要进行服务器来进行特征的抽取，所以我们需要很将单张图片丢入caffe的网络进行一次传递，这样就诞生了一个从内存中如何加载数据进入caffe的需求，这里我直接贴出代码来先：

**[cpp]** [view plaincopy](http://blog.csdn.net/cparent/article/details/47008485)

1. #include <boost/make\_shared.hpp>

4. // these need to be included after boost on OS X
5. #include <string>  // NOLINT(build/include\_order)
6. #include <vector>  // NOLINT(build/include\_order)
7. #include <fstream>  // NOLINT

10. #include "caffe/caffe.hpp"
11. #include <opencv.hpp>

14. **static** **void** CheckFile(**const** std::string& filename) {
15. std::ifstream f(filename.c\_str());
16. **if** (!f.good()) {
17. f.close();
18. **throw** std::runtime\_error("Could not open file " + filename);
19. }
20. f.close();
21. }



26. **template** <**typename** Dtype>
27. caffe::Net<Dtype>\* Net\_Init\_Load(
28. std::string param\_file, std::string pretrained\_param\_file, caffe::Phase phase)
29. {
30. CheckFile(param\_file);
31. CheckFile(pretrained\_param\_file);

34. caffe::Net<Dtype>\* net(**new** caffe::Net<Dtype>(param\_file,phase));


38. net->CopyTrainedLayersFrom(pretrained\_param\_file,0);
39. **return** net;
40. }
41. #define NetF float



46. **int** main()
47. {
48. cv::Mat src1;
49. src1 = cv::imread("test.png");

52. cv::Mat rszimage;

55. //// The mean file image size is 256x256, need to resize the input image to 256x256
56. cv::resize(src1, rszimage, cv::Size(227, 227));
57. std::vector<cv::Mat> dv = { rszimage }; // image is a cv::Mat, as I'm using #1416
58. std::vector<**int**> dvl = { 0 };
60. caffe::Datum data;
61. caffe::ReadFileToDatum("D:/work/DestImage/crop/CH0005-00-0019/00028.png", &data);

64. caffe::Net<NetF>\* \_net = Net\_Init\_Load<NetF>("deploy\_Test.prototxt", "bvlc\_alexnet.caffemodel", caffe::TEST);
65. caffe::MemoryDataLayer<NetF> \*m\_layer\_ = (caffe::MemoryDataLayer<NetF> \*)\_net->layers()[0].get();
66. m\_layer\_->AddMatVector(dv, dvl);
68. /\*float loss = 0.0;
69. std::vector<caffe::Blob<float>\*> results = \_net->ForwardPrefilled(&loss);\*/
70. **int** end\_ind = \_net->layers().size();
71. std::vector<caffe::Blob<NetF>\*> input\_vec;
72. \_net->Forward(input\_vec);
73. boost::shared\_ptr<caffe::Blob<NetF>> outPool5 = \_net->blob\_by\_name("pool5");
74. std::cout << outPool5->shape()[0] << std::endl;
75. std::cout << outPool5->shape()[1] << std::endl;
76. std::cout << outPool5->shape()[2] << std::endl;
77. std::cout << outPool5->shape()[3] << std::endl;

80. std::cout << outPool5->num() << std::endl;
81. std::cout << outPool5->channels() << std::endl;
82. std::cout << outPool5->width() << std::endl;
83. std::cout << outPool5->height() << std::endl;
84. std::cout << outPool5->data\_at(0, 0, 0, 0) << std::endl;
85. std::cout << outPool5->data\_at(0, 0, 1, 1) << std::endl;
86. std::cout << outPool5->data\_at(0, 95, 5, 5) << std::endl;

89. **const** NetF\* pstart = outPool5->cpu\_data();
90. std::cout << m\_layer\_->width() << std::endl;
92. **return** 0;
93. }

然后是配置文件：

**[plain]** [view plaincopy](http://blog.csdn.net/cparent/article/details/47008485)

1. name: "CaffeNet"

4. layers
5. {
6. name: "data"
7. type: MEMORY\_DATA
8. top: "data"
9. top: "label"
10. memory\_data\_param
11. {
12. batch\_size: 1
13. channels: 3
14. height: 227
15. width: 227
16. }
17. transform\_param
18. {
19. crop\_size: 227
20. mirror: false
21. #mean\_file:"imagenet\_mean.binaryproto"
22. mean\_value: 104
23. mean\_value: 117
24. mean\_value: 123
25. }
26. }

29. layers {
30. name: "`"
31. type: CONVOLUTION
32. bottom: "data"
33. top: "conv1"
34. blobs\_lr: 1
35. blobs\_lr: 2
36. weight\_decay: 1
37. weight\_decay: 0
38. convolution\_param {
39. num\_output: 96
40. kernel\_size: 11
41. stride: 4
42. }
43. }
44. layers {
45. name: "relu1"
46. type: RELU
47. bottom: "conv1"
48. top: "conv1"
49. }
50. layers {
51. name: "pool1"
52. type: POOLING
53. bottom: "conv1"
54. top: "pool1"
55. pooling\_param {
56. pool: MAX
57. kernel\_size: 3
58. stride: 2
59. }
60. }
61. layers {
62. name: "norm1"
63. type: LRN
64. bottom: "pool1"
65. top: "norm1"
66. lrn\_param {
67. local\_size: 5
68. alpha: 0.0001
69. beta: 0.75
70. }
71. }
72. layers {
73. name: "conv2"
74. type: CONVOLUTION
75. bottom: "norm1"
76. top: "conv2"
77. blobs\_lr: 1
78. blobs\_lr: 2
79. weight\_decay: 1
80. weight\_decay: 0
81. convolution\_param {
82. num\_output: 256
83. pad: 2
84. kernel\_size: 5
85. group: 2
86. }
87. }
88. layers {
89. name: "relu2"
90. type: RELU
91. bottom: "conv2"
92. top: "conv2"
93. }
94. layers {
95. name: "pool2"
96. type: POOLING
97. bottom: "conv2"
98. top: "pool2"
99. pooling\_param {
100. pool: MAX
101. kernel\_size: 3
102. stride: 2
103. }
104. }
105. layers {
106. name: "norm2"
107. type: LRN
108. bottom: "pool2"
109. top: "norm2"
110. lrn\_param {
111. local\_size: 5
112. alpha: 0.0001
113. beta: 0.75
114. }
115. }
116. layers {
117. name: "conv3"
118. type: CONVOLUTION
119. bottom: "norm2"
120. top: "conv3"
121. blobs\_lr: 1
122. blobs\_lr: 2
123. weight\_decay: 1
124. weight\_decay: 0
125. convolution\_param {
126. num\_output: 384
127. pad: 1
128. kernel\_size: 3
129. }
130. }
131. layers {
132. name: "relu3"
133. type: RELU
134. bottom: "conv3"
135. top: "conv3"
136. }
137. layers {
138. name: "conv4"
139. type: CONVOLUTION
140. bottom: "conv3"
141. top: "conv4"
142. blobs\_lr: 1
143. blobs\_lr: 2
144. weight\_decay: 1
145. weight\_decay: 0
146. convolution\_param {
147. num\_output: 384
148. pad: 1
149. kernel\_size: 3
150. group: 2
151. }
152. }
153. layers {
154. name: "relu4"
155. type: RELU
156. bottom: "conv4"
157. top: "conv4"
158. }
159. layers {
160. name: "conv5"
161. type: CONVOLUTION
162. bottom: "conv4"
163. top: "conv5"
164. blobs\_lr: 1
165. blobs\_lr: 2
166. weight\_decay: 1
167. weight\_decay: 0
168. convolution\_param {
169. num\_output: 256
170. pad: 1
171. kernel\_size: 3
172. group: 2
173. }
174. }
175. layers {
176. name: "relu5"
177. type: RELU
178. bottom: "conv5"
179. top: "conv5"
180. }
181. layers {
182. name: "pool5"
183. type: POOLING
184. bottom: "conv5"
185. top: "pool5"
186. pooling\_param {
187. pool: MAX
188. kernel\_size: 3
189. stride: 2
190. }
191. }
192. layers {
193. name: "fc6"
194. type: INNER\_PRODUCT
195. bottom: "pool5"
196. top: "fc6"
197. blobs\_lr: 1
198. blobs\_lr: 2
199. weight\_decay: 1
200. weight\_decay: 0
201. inner\_product\_param {
202. num\_output: 4096
203. }
204. }
205. layers {
206. name: "relu6"
207. type: RELU
208. bottom: "fc6"
209. top: "fc6"
210. }
211. layers {
212. name: "drop6"
213. type: DROPOUT
214. bottom: "fc6"
215. top: "fc6"
216. dropout\_param {
217. dropout\_ratio: 0.5
218. }
219. }
220. layers {
221. name: "fc7"
222. type: INNER\_PRODUCT
223. bottom: "fc6"
224. top: "fc7"
225. blobs\_lr: 1
226. blobs\_lr: 2
227. weight\_decay: 1
228. weight\_decay: 0
229. inner\_product\_param {
230. num\_output: 4096
231. }
232. }
233. layers {
234. name: "relu7"
235. type: RELU
236. bottom: "fc7"
237. top: "fc7"
238. }
239. layers {
240. name: "drop7"
241. type: DROPOUT
242. bottom: "fc7"
243. top: "fc7"
244. dropout\_param {
245. dropout\_ratio: 0.5
246. }
247. }
248. layers {
249. name: "fc8"
250. type: INNER\_PRODUCT
251. bottom: "fc7"
252. top: "fc8"
253. blobs\_lr: 1
254. blobs\_lr: 2
255. weight\_decay: 1
256. weight\_decay: 0
257. inner\_product\_param {
258. num\_output: 1000
259. }
260. }
262. layers
263. {
264. name: "prob"
265. type: SOFTMAX
266. bottom: "fc8"
267. top: "prob"
268. }
270. layers
271. {
272. name: "output"
273. type: ARGMAX
274. bottom: "prob"
275. top: "output"
276. }

我的模型使用的是alexnet，例子是用来抽取一个图片在pool5那一层的特征。这样大家使用这个例子可以利用caffe的任意模型抽取任意图片的特征。