# [deep learning实践经验总结](http://blog.csdn.net/lingerlanlan/article/details/31773811)

分类： [深度学习（deep learning）](http://blog.csdn.net/linger2012liu/article/category/2146595)2014-06-17 19:18 2109人阅读 [评论](http://blog.csdn.net/lingerlanlan/article/details/31773811#comments)(1) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/lingerlanlan/article/details/31773811#report)

[深度学习](http://www.csdn.net/tag/%e6%b7%b1%e5%ba%a6%e5%ad%a6%e4%b9%a0)[神经网络](http://www.csdn.net/tag/%e7%a5%9e%e7%bb%8f%e7%bd%91%e7%bb%9c)[caffe](http://www.csdn.net/tag/caffe)[deep learning](http://www.csdn.net/tag/deep%20learning)

最近拿caffe来做图片分类，遇到不少问题，同时也吸取不少教训和获得不少经验。

先看例子再总结经验。

这是一个2类分类器，分的是条纹衣服和纯色衣服。

先看几张图片。

条纹衣服：

纯色衣服:

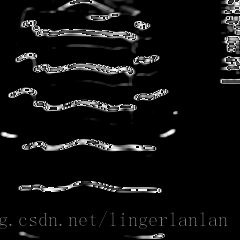


肉眼也很容易辨认出来。

训练出来的模型目前的准确率是0.75。

为了可视化特征抽取，我把某一层的特征图和权重图也画出来了，这层是其中一个全连接层。

条纹衣服的特征图：

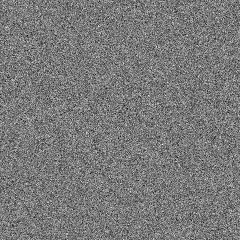
    

纯色衣服的特征图：

看条纹衣服的特征图比较有意思，把“条纹”特征给抽取出来了。也许这就是神经网络神奇的地方，在没有人的干扰的情况下，竟然能学习出来“条纹”特征。

其中一个channel的权重图：



这个就看不出来什么了。曾经有一个数据集，训练的是裙子的模型，当我看到权重图是一个裙子轮廓的图。

好了，说了这么多，总结一下经验吧。

1 数据集要保证质量。曾经玩过一字领和polo领的分类，刚开始效果很差，后来发现有一些“错误”的标签，于是把那些样本给去掉。效果好了很多。

2 learning rate要调整。有一次训练了很久，准确率几乎不变，于是我减少了lr，发现好了很多。

3 均值化图片。实践证明，均值化后再训练收敛速度更快，准确率更高。

对于深度学习的困惑：感觉准确率是个大问题啊，其实这也是所有机器学习算法的通病。

对于别人研究提供的数据集，比如imagenet,cifa10,lenet，效果很好。

但是自己收集的数据集，效果就不是很理想了。

也就是说，算法不是万能的，只是对于某些数据集有效。

我们能做的，是什么？

1 对于哪些数据集，深度学习比较适合？

2 对于效果差的数据集，如何能提高准确率？

曾经脑海里闪过一个念头，是由上面提到的权重图想到的。

当时看到权重图是一个裙子轮廓的图，心里就想。

这是神经网络自动调整出来的权重图，

如果人为加上干预，能否实现优化呢？

# [deep learning实践经验总结2--准确率再次提升，到达0.8，再来总结一下](http://blog.csdn.net/lingerlanlan/article/details/32329761)

分类： [深度学习（deep learning）](http://blog.csdn.net/linger2012liu/article/category/2146595)2014-06-19 15:25 4112人阅读 [评论](http://blog.csdn.net/lingerlanlan/article/details/32329761#comments)(15) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/lingerlanlan/article/details/32329761#report)

[deep learning](http://www.csdn.net/tag/deep%20learning)[机器学习](http://www.csdn.net/tag/%e6%9c%ba%e5%99%a8%e5%ad%a6%e4%b9%a0)[caffe](http://www.csdn.net/tag/caffe)[图像处理](http://www.csdn.net/tag/%e5%9b%be%e5%83%8f%e5%a4%84%e7%90%86)

deep learning实践经验总结2

最近拿caffe来做图片分类，遇到不少问题，同时也吸取不少教训和获得不少经验。

这次拿大摆裙和一步裙做分类，

多次训练效果一直在0.7，后来改动了全链接层的初始化参数。高斯分布的标准差由0.001改为0.0001，就是调小了。

然后效果很明显，准确率高了，权重图画出来后，也看得出是有意义的了，部分权重图是人的轮廓或者裙子的轮廓。

先看看图片：

大摆裙

一步裙

然后找一些响应图看一下，当然我这里展示的是一些效果好的响应图。

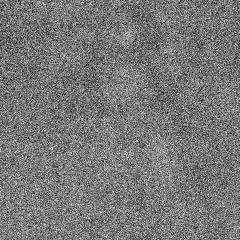
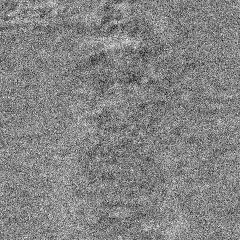
大摆裙

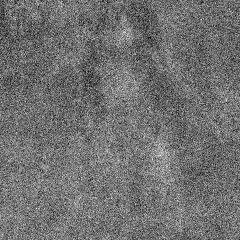
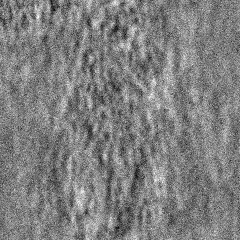
     

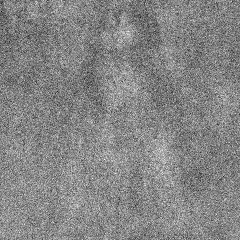
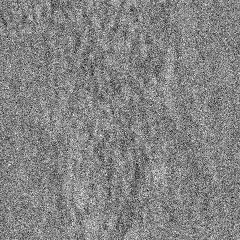
一步裙

一些权重图：

这是网络的结构参数：

**[ruby]** [view plaincopy](http://blog.csdn.net/lingerlanlan/article/details/32329761)

1. name: "CIFAR10\_full\_train"
2. layers {
3. layer {
4. name: "cifar"
5. type: "data"
6. #source: "/home/linger/linger/testfile/crop\_train\_db"
7. #source: "/home/linger/linger/testfile/collar\_train\_db"
8. source: "/home/linger/linger/testfile/skirt\_train\_db"
9. #source: "/home/linger/linger/testfile/pattern\_train\_db"
10. meanfile: "/home/linger/linger/testfile/skirt\_train\_mean.binaryproto"
11. #cropsize: 200
12. batchsize: 20
13. }
14. top: "data"
15. top: "label"
16. }
18. layers {
19. layer {
20. name: "conv1"
21. type: "conv"
22. num\_output: 16
23. kernelsize: 5
24. stride:1
25. weight\_filler {
26. type: "gaussian"
27. std: 0.01
28. }
29. bias\_filler {
30. type: "constant"
31. value: 0.
32. }
33. blobs\_lr: 1.
34. blobs\_lr: 1.
35. weight\_decay: 0.001
36. weight\_decay: 0.
37. }
38. bottom: "data"
39. top: "conv1"
40. }
41. layers {
42. layer {
43. name: "relu1"
44. type: "relu"
45. }
46. bottom: "conv1"
47. top: "conv1"
48. }
49. layers {
50. layer {
51. name: "pool1"
52. type: "pool"
53. pool: MAX
54. kernelsize: 2
55. stride:1
56. }
57. bottom: "conv1"
58. top: "pool1"
59. }
60. layers {
61. layer {
62. name: "conv2"
63. type: "conv"
64. num\_output: 16
65. group: 2
66. kernelsize: 5
67. stride:1
68. weight\_filler {
69. type: "gaussian"
70. std: 0.01
71. }
72. bias\_filler {
73. type: "constant"
74. value: 0.
75. }
76. blobs\_lr: 1.
77. blobs\_lr: 1.
78. weight\_decay: 0.001
79. weight\_decay: 0.
80. }
81. bottom: "pool1"
82. top: "conv2"
83. }
84. layers {
85. layer {
86. name: "relu2"
87. type: "relu"
88. }
89. bottom: "conv2"
90. top: "conv2"
91. }
92. layers {
93. layer {
94. name: "pool2"
95. type: "pool"
96. pool: MAX
97. kernelsize: 2
98. stride: 1
99. }
100. bottom: "conv2"
101. top: "pool2"
102. }
104. layers {
105. layer {
106. name: "ip1"
107. type: "innerproduct"
108. num\_output: 100
109. weight\_filler {
110. type: "gaussian"
111. std: 0.0001
112. }
113. bias\_filler {
114. type: "constant"
115. value: 0.
116. }
117. blobs\_lr: 1.
118. blobs\_lr: 1.
119. weight\_decay: 0.001
120. weight\_decay: 0.
121. }
122. bottom: "pool2"
123. top: "ip1"
124. }
126. layers {
127. layer {
128. name: "ip2"
129. type: "innerproduct"
130. num\_output: 2
131. weight\_filler {
132. type: "gaussian"
133. std: 0.01
134. }
135. bias\_filler {
136. type: "constant"
137. value: 0.
138. }
139. blobs\_lr: 1.
140. blobs\_lr: 1.
141. weight\_decay: 0.001
142. weight\_decay: 0.
143. }
144. bottom: "ip1"
145. top: "ip2"
146. }
148. #-----------------------output------------------------
149. layers {
150. layer {
151. name: "loss"
152. type: "softmax\_loss"
153. }
154. bottom: "ip2"
155. bottom: "label"
156. }

**[ruby]** [view plaincopy](http://blog.csdn.net/lingerlanlan/article/details/32329761)

1. name: "CIFAR10\_full\_test"
2. layers {
3. layer {
4. name: "cifar"
5. type: "data"
6. #source: "/home/linger/linger/testfile/collar\_test\_db"
7. #source: "/home/linger/linger/testfile/crop\_test\_db"
8. source: "/home/linger/linger/testfile/skirt\_test\_db"
9. #source: "/home/linger/linger/testfile/pattern\_test\_db"
10. meanfile: "/home/linger/linger/testfile/skirt\_test\_mean.binaryproto"
11. #cropsize: 200
12. batchsize: 10
13. }
14. top: "data"
15. top: "label"
16. }
18. layers {
19. layer {
20. name: "conv1"
21. type: "conv"
22. num\_output: 16
23. kernelsize: 5
24. stride:1
25. weight\_filler {
26. type: "gaussian"
27. std: 0.01
28. }
29. bias\_filler {
30. type: "constant"
31. value: 0.
32. }
33. blobs\_lr: 1.
34. blobs\_lr: 1.
35. weight\_decay: 0.001
36. weight\_decay: 0.
37. }
38. bottom: "data"
39. top: "conv1"
40. }
41. layers {
42. layer {
43. name: "relu1"
44. type: "relu"
45. }
46. bottom: "conv1"
47. top: "conv1"
48. }
49. layers {
50. layer {
51. name: "pool1"
52. type: "pool"
53. pool: MAX
54. kernelsize: 2
55. stride:1
56. }
57. bottom: "conv1"
58. top: "pool1"
59. }
60. layers {
61. layer {
62. name: "conv2"
63. type: "conv"
64. num\_output: 16
65. group: 2
66. kernelsize: 5
67. stride:1
68. weight\_filler {
69. type: "gaussian"
70. std: 0.01
71. }
72. bias\_filler {
73. type: "constant"
74. value: 0.
75. }
76. blobs\_lr: 1.
77. blobs\_lr: 1.
78. weight\_decay: 0.001
79. weight\_decay: 0.
80. }
81. bottom: "pool1"
82. top: "conv2"
83. }
84. layers {
85. layer {
86. name: "relu2"
87. type: "relu"
88. }
89. bottom: "conv2"
90. top: "conv2"
91. }
92. layers {
93. layer {
94. name: "pool2"
95. type: "pool"
96. pool: MAX
97. kernelsize: 2
98. stride: 1
99. }
100. bottom: "conv2"
101. top: "pool2"
102. }
103. layers {
104. layer {
105. name: "ip1"
106. type: "innerproduct"
107. num\_output: 100
108. weight\_filler {
109. type: "gaussian"
110. std: 0.0001
111. }
112. bias\_filler {
113. type: "constant"
114. value: 0.
115. }
116. blobs\_lr: 1.
117. blobs\_lr: 1.
118. weight\_decay: 0.001
119. weight\_decay: 0.
120. }
121. bottom: "pool2"
122. top: "ip1"
123. }
125. layers {
126. layer {
127. name: "ip2"
128. type: "innerproduct"
129. num\_output: 2
130. weight\_filler {
131. type: "gaussian"
132. std: 0.01
133. }
134. bias\_filler {
135. type: "constant"
136. value: 0.
137. }
138. blobs\_lr: 1.
139. blobs\_lr: 1.
140. weight\_decay: 0.001
141. weight\_decay: 0.
142. }
143. bottom: "ip1"
144. top: "ip2"
145. }
147. #-----------------------output------------------------
148. layers {
149. layer {
150. name: "prob"
151. type: "softmax"
152. }
153. bottom: "ip2"
154. top: "prob"
155. }
156. layers {
157. layer {
158. name: "accuracy"
159. type: "accuracy"
160. }
161. bottom: "prob"
162. bottom: "label"
163. top: "accuracy"
164. }

**[ruby]** [view plaincopy](http://blog.csdn.net/lingerlanlan/article/details/32329761)

1. # reduce learning rate after 120 epochs (60000 iters) by factor 0f 10
2. # then another factor of 10 after 10 more epochs (5000 iters)
4. # The training protocol buffer definition
5. train\_net: "cifar10\_full\_train.prototxt"
6. # The testing protocol buffer definition
7. test\_net: "cifar10\_full\_test.prototxt"
8. # test\_iter specifies how many forward passes the test should carry out.
9. # In the case of CIFAR10, we have test batch size 100 and 100 test iterations,
10. # covering the full 10,000 testing images.
11. test\_iter: 20
12. # Carry out testing every 1000 training iterations.
13. test\_interval: 100
14. # The base learning rate, momentum and the weight decay of the network.
15. base\_lr: 0.00001
16. momentum: 0.9
17. weight\_decay: 0.004
18. # The learning rate policy
19. lr\_policy: "fixed"
20. # Display every 200 iterations
21. display: 20
22. # The maximum number of iterations
23. max\_iter: 60000
24. # snapshot intermediate results
25. snapshot: 1000
26. snapshot\_prefix: "cifar10\_full"
27. # solver mode: 0 for CPU and 1 for GPU
28. solver\_mode: 1

真的是多玩数据，才会对数据形成一种感觉啊。

下次玩3类的。敬请期待！

# [神经网络：caffe特征可视化的代码样例](http://blog.csdn.net/lingerlanlan/article/details/37593837)

分类： [深度学习（deep learning）](http://blog.csdn.net/linger2012liu/article/category/2146595)2014-07-09 12:56 4554人阅读 [评论](http://blog.csdn.net/lingerlanlan/article/details/37593837#comments)(28) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/lingerlanlan/article/details/37593837#report)

[caffe](http://www.csdn.net/tag/caffe)[神经网络](http://www.csdn.net/tag/%e7%a5%9e%e7%bb%8f%e7%bd%91%e7%bb%9c)[深度学习](http://www.csdn.net/tag/%e6%b7%b1%e5%ba%a6%e5%ad%a6%e4%b9%a0)[cuda](http://www.csdn.net/tag/cuda)[deep learning](http://www.csdn.net/tag/deep%20learning)

caffe特征可视化的代码样例

不少读者看了我前面两篇文章

[总结一下用caffe跑图片数据的研究流程](http://blog.csdn.net/lingerlanlan/article/details/31763799)

[deep learning实践经验总结2--准确率再次提升，到达0.8，再来总结一下](http://blog.csdn.net/lingerlanlan/article/details/32329761)

之后，想知道我是怎么实现特征可视化的。

简单来说，其实就是让神经网络正向传播一次，然后把某层的特征值给取出来，然后转换为图片保存。

下面我提供一个demo，大家可以根据自己的需求修改。

先看看我的demo的使用方法。

visualize\_features.bin net\_proto pretrained\_net\_proto iterations  [CPU/GPU]  img\_list\_file dstdir laydepth

visualize\_features.bin是cpp编译出来的可执行文件

下面看看各参数的意义：

1 net\_proto：caffe规定的一种定义网络结构的文件格式，后缀名为".prototxt"。这个文件定义了网络的输入，已经相关参数，还有就是整体的网络结构。

2 pretrained\_net\_proto：这个是已经训练好了的模型

3 iterations：迭代次数

4 [CPU/GPU]：cpu还是gpu模式

5 img\_list\_file：待测试的文件名列表。我这里需要这个主要是为了得到图片的类名。

6 dstdir：图片输出的文件夹

7 laydepth：需要输出哪一层的特征

下面是一个实例例子：

./visualize\_features.bin /home/linger/linger/caffe-action/caffe-master/examples/cifar10/cifar10\_full\_test.prototxt /home/linger/linger/caffe-action/caffe-master/examples/cifar10/cifar10\_full\_iter\_60000 20 GPU /home/linger/linger/testfile/skirt\_test\_attachment/image\_filename /home/linger/linger/testfile/innerproduct/ 7

下面是源代码：

**[cpp]** [view plaincopy](http://blog.csdn.net/lingerlanlan/article/details/37593837)

1. // Copyright 2013 Yangqing Jia
2. //
3. // This is a simple script that allows one to quickly test a network whose
4. // structure is specified by text format protocol buffers, and whose parameter
5. // are loaded from a pre-trained network.
6. // Usage:
7. //    test\_net net\_proto pretrained\_net\_proto iterations [CPU/GPU]
9. #include <cuda\_runtime.h>
10. #include <fstream>
11. #include <iostream>
12. #include <cstring>
13. #include <cstdlib>
14. #include <algorithm>
15. #include <vector>
16. #include <utility>
17. #include "caffe/caffe.hpp"
18. #include <opencv2/highgui/highgui.hpp>
19. #include <opencv2/highgui/highgui\_c.h>
20. #include <opencv2/imgproc/imgproc.hpp>
22. **using** std::make\_pair;
23. **using** std::pair;
24. **using** **namespace** caffe;  // NOLINT(build/namespaces)
25. **using** **namespace** std;
27. vector<string> fileNames;
28. **char** \* filelist;
30. /\*
31. \* 读入的文件的内容格式类似这样子的：全局id 类名\_所在类的id.jpg
32. 0 一步裙\_0.jpg
33. 1 一步裙\_1.jpg
34. 2 一步裙\_10.jpg
35. \*/
36. **void** readFile()
37. {
38. **if**(fileNames.empty())
39. {
40. ifstream read(filelist);
41. //"/home/linger/linger/testfile/test\_attachment/image\_filename"
42. // "/home/linger/imdata/test\_files\_collar.txt"
43. //  "/home/linger/linger/testfile/testfilename"
44. **if**(read.is\_open())
45. {
46. **while**(!read.eof())
47. {
48. string name;
49. **int** id;
50. read>>id>>name;
51. fileNames.push\_back(name);
52. }
53. }
54. }
55. }
57. /\*
58. \* 根据图片id获取类名
59. \*/
60. string getClassNameById(**int** id)
61. {
62. readFile();
63. **int** index = fileNames[id].find\_last\_of('\_') ;
64. **return** fileNames[id].substr(0, index);
65. }


69. **void** writeBatch(**const** **float**\* data,**int** num,**int** channels,**int** width,**int** height,**int** startID,**const** **char**\*dir)
70. {
71. **for**(**int** id = 0;id<num;id++)
72. {
73. **for**(**int** channel=0;channel<channels;channel++)
74. {
75. cv::Mat mat(height,width, CV\_8UC1);//高宽
76. vector<vector<**float**> > vec;
77. vec.resize(height);
78. **float** max = -1;
79. **float** min = 999999;
80. **for**(**int** row=0;row<height;row++)
81. {
82. vec[row].resize(width);
83. **for**(**int** col=0;col<width;col++)
84. {
85. vec[row][col] =
86. data[id\*channels\*width\*height+channel\*width\*height+row\*width+col];
87. **if**(max<vec[row][col])
88. {
89. max = vec[row][col];
90. }
91. **if**(min>vec[row][col])
92. {
93. min = vec[row][col];
94. }

97. }
98. }
100. **for**(**int** row=0;row<height;row++)
101. {
102. **for**(**int** col=0;col<width;col++)
103. {
104. vec[row][col] = 255\*((**float**)(vec[row][col]-min))/(max-min);
105. uchar& img = mat.at<uchar>(row,col);
106. img= vec[row][col];
108. }
109. }
110. **char** filename[100];
111. string label = getClassNameById(startID+id);
112. string file\_reg =dir;
113. file\_reg+="%s%05d\_%05d.png";
114. snprintf(filename, 100, file\_reg.c\_str(), label.c\_str(),startID+id,channel);
115. //printf("%s\n",filename);
116. cv::imwrite(filename, mat);
117. }
119. }
120. }
122. **int** main(**int** argc, **char**\*\* argv)
123. {
124. **if** (argc < 4)
125. {
126. LOG(ERROR) << "visualize\_features.bin net\_proto pretrained\_net\_proto iterations "
127. << "[CPU/GPU] img\_list\_file dstdir laydepth";
128. **return** 0;
129. }
130. /\*
132. ./visualize\_features.bin /home/linger/linger/caffe-action/caffee-ext/Caffe\_MM/prototxt/triplet/triplet\_test\_simple.prototxt /home/linger/linger/caffe-action/caffee-ext/Caffe\_MM/snapshorts/\_iter\_100000 8 GPU /home/linger/linger/testfile/test\_attachment/image\_filename /home/linger/linger/testfile/innerproduct/ 6
134. \*/
136. filelist = argv[5];
137. cudaSetDevice(0);
138. Caffe::set\_phase(Caffe::TEST);
140. **if** (argc == 5 && strcmp(argv[4], "GPU") == 0)
141. {
142. LOG(ERROR) << "Using GPU";
143. Caffe::set\_mode(Caffe::GPU);
144. }
145. **else**
146. {
147. LOG(ERROR) << "Using CPU";
148. Caffe::set\_mode(Caffe::CPU);
149. }
151. NetParameter test\_net\_param;
152. ReadProtoFromTextFile(argv[1], &test\_net\_param);
153. Net<**float**> caffe\_test\_net(test\_net\_param);
154. NetParameter trained\_net\_param;
155. ReadProtoFromBinaryFile(argv[2], &trained\_net\_param);
156. caffe\_test\_net.CopyTrainedLayersFrom(trained\_net\_param);
158. **int** total\_iter = atoi(argv[3]);
159. LOG(ERROR) << "Running " << total\_iter << " Iterations.";
161. **double** test\_accuracy = 0;
162. vector<Blob<**float**>\*> dummy\_blob\_input\_vec;
164. **int** startID = 0;
165. **int** nums;
166. **int** dims;
167. **int** batchsize = test\_net\_param.layers(0).layer().batchsize();
169. **int** laynum = caffe\_test\_net.bottom\_vecs().size();
170. printf("num of layers:%d\n",laynum);
172. **for** (**int** i = 0; i < total\_iter; ++i)
173. {
174. **const** vector<Blob<**float**>\*>& result =
175. caffe\_test\_net.Forward(dummy\_blob\_input\_vec);
177. **int** laydepth = atoi(argv[7]);
179. Blob<**float**>\* features = (\*(caffe\_test\_net.bottom\_vecs().begin()+laydepth))[0];//调整第几层即可
181. nums = features->num();
182. dims= features->count()/features->num();
184. **int** num = features->num();
185. **int** channels = features->channels();
186. **int** width = features->width();
187. **int** height = features->height();
188. printf("channels:%d,width:%d,height:%d\n",channels,width,height);
189. writeBatch(features->cpu\_data(),num,channels,width,height,startID,argv[6]);
190. startID += nums;
192. }
194. **return** 0;
195. }