[使用Caffe对图片进行训练并分类的简单流程](http://blog.csdn.net/deeplearninglc007/article/details/40086503)

**step 1.**  首先，确保caffe已经正确安装，并且make runtest基本通过。

**step 2.**   准备训练集：

        在训练之前，要准备训练需要的训练集，为了验证训练的效果，最好也准备一定数量的测试集。由于caffe的数据集的输入是leveldb格式，因此在训练前需要将输入转为相应格式。

        在caffe\_root/example 文件夹中，提供了一些例子，cifar10与imagenet均是将图片形式数据库转换为leveldb格式，mnist则是将其本身的数据集转换为leveldb格式。这就要求我们需要把自己的数据集转换成leveldb格式，需要自己编写程序实现。下面以jpg格式图片为例说明。

在此，假设数据库本身是以图片形式给出，并且给出了label或者同一类别的图片已经分类。这样我们就可以通过imagenet例子中给出的create\_imagenet.sh 对我们自己的数据库进行转换，要求数据集图片格式为jpg，以下为具体操作步骤：

        A．若数据集已经给出label则忽略此步骤。图片按照类别装在不同文件夹下的情况，自己编写mklabel.sh命令，对图片进行处理并标明label。具体操作参照mklabel.sh 程序说明。

**[cpp]** [view plaincopy](http://blog.csdn.net/deeplearninglc007/article/details/40086503)

1. <pre name="code" **class**="plain">mklabel.sh
2. #!/bin/sh
4. #----------------------------------------------------
5. #文件存放形式为
6. #   dir/subdir1/files...
7. #   dir/subdir2/files...
8. #   dir/subdir3/files...
9. #   dir/subdirX/files...
11. #用法：
12. #1.$ sh mklabel.sh dir startlabel ;dir 为目标文件夹名称
13. #2.$ chmod a+x mklabel.sh ；然后可以直接用文件名运行
14. #3.默认label信息显示在终端，请使用转向符'>'生成文本，例：
15. #       $ sh ./mklabel.sh  data/faces94/male  > label.txt
16. #4.确保文件夹下除了图片不含其他文件(若含有则需自行添加判断语句)
17. #-----------------------------------------------------
19. DIR=~/codes/mklabel.sh      #命令位置（无用）
20. label=1                 #label起始编号(为数字，根据自己需要修改)
21. testnum=0               #保留的测试集大小
23. **if** test $# -eq 0;then   #无参数，默认为当前文件夹下，label=1
24. $DIR . 0 $label
25. **else**
26. **if** test $# -eq 1;then   #仅有位置参数，默认testnum=0,label=1
27. $DIR $1 0 $label
28. **else**
29. **if** test $# -eq 2;then   #两个参数时,label=1
30. $DIR $1 $2 $label
31. **else**
32. testnum=$2          #每个类别保留测试集大小
33. label=$3            #自定义label起始
35. cd $1               #转到目标文件夹
37. **if** test $testnum -ne 0;then
38. mkdir "testdata"    #建立测试集
39. fi
41. **for** i in \* ; **do**
42. exist=`expr "$i" != "testdata"`
43. **if** test -d $i && test $exist -eq 1;then #文件夹存在
44. #echo
45. #echo 'DIR:' $i
47. cd $i           #进入文件夹
48. num=1       #图片数目
49. **for** j in \*
50. **do**
51. **if** test $num -gt $testnum;then
52. echo  $j  $label
53. mv $j ../
54. fi
55. num=`expr $num + 1`
56. done
57. cd ..           #回到上层目录
59. **if** test $testnum -eq 0;then
60. rmdir $i
61. **else**
62. mv $i ./testdata
63. fi
65. label=`expr $label + 1`
66. #计算label
67. fi
68. done
69. fi
70. fi
71. fi

        B．修改create\_imagenet.sh 文件，使其中的图片源地址与标明label的txt文件对应到自己数据库的相应文件。其中的convert\_imageset.cpp 函数的参数依次为图片文件夹位置，label.txt文本文件，目标数据集名称，是否随机存储图片与label（即打乱图片数据的读入顺序）。

若你所使用的数据集不是通过图片形式给出，为了方便，可以根据leveldb数据的存储格式，自己编写程序转换数据集。

        C．多通道图片需要用到meanfile，通过example/imagenet文件夹下的shell函数make\_imagenet\_mean.sh，更改相应函数，很轻松的得到binaryproto文件，在训练时可能需要用到。

**step 3.**  使用自己的数据进行训练：

        以最简单的mnist网络为例，因为数据集的不同则需要更改的参数也不同。在训练前最好在example下新建个文件夹命名为数据集的名称，然后把mnist下的5个文件：

        lenet.prototxt

        lenet\_solver.prototxt

        lenet\_train.prototxt

        lenet\_test.prototxt

        train\_lenet.sh

      复制到新文件夹下，以上5个文件为必需的文件。按顺序对以上文件进行修改，在忽略网络结构的情况下，需要修改的有:

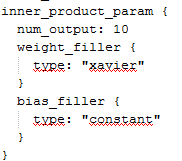
        a.      lenet.prototxt：

**[plain]** [view plaincopy](http://blog.csdn.net/deeplearninglc007/article/details/40086503)

1. input\_dim: 64
2. input\_dim: 1
3. input\_dim: 28
4. input\_dim: 28

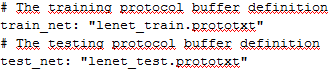
        分别为一次导入的图片个数，channel，heigth ，width。

                      倒数第二层，即输入给softmax层数据的那层，其中的num\_output需要更改为实际图片的label数，即图片类别数。否则在进行训练时，会出现错误。



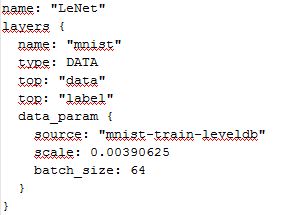
        b.      lenet\_solver.prototxt：

       如果之前文件名没改的话则不需要更改以上两项，否则改为对应文件。其他参数根据实际需要更改。

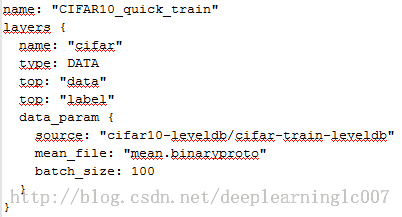


        c.      lenet\_train.prototxt：

        需要把data层的数据源文件替换为自己的数据。



        在训练多通道图片时，此处最好需要有一个meanfile参数。例如cifar10



       num\_output参数参照lenet.prototxt修改。

        d.      lenet\_test.prototxt：

        参照lenet\_train.prototxt 进行相应修改。

        e.      train\_lenet.sh：

         lenet\_solver.prototxt文件名未更改的话则不需要进行改动。

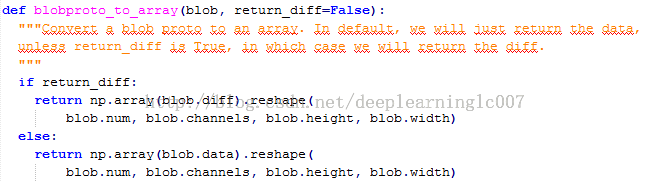
**step 4.**  使用自己的model进行分类：

        Reference: <http://nbviewer.ipython.org/github/BVLC/caffe/blob/master/examples/classification.ipynb>

         假设之前的所有步骤已经成功，并且已经训练出正确率还行的model，则现在就可以使用model对图片进行分类。

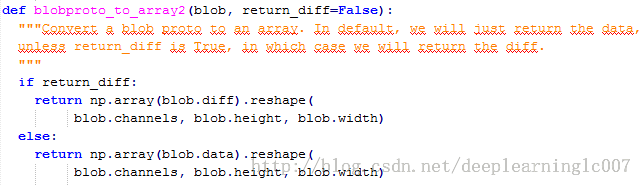
        a.      首先确保已经正确安装了pythonwrapper，以及caffe\_root/python/文件夹下requirements.txt文件中所需要的组件。

        b.      另外，还需要meanfile的npy文件。即需要把数据对应的binaryproto文件转换到npy文件。Caffe\_root/python/caffe/io.cpp文件中已经给了对应的API。

****

        具体参照：<https://github.com/BVLC/caffe/issues/420>

        需要对blobproto\_to\_array 进行修改，得到blobproto\_to\_array2.



        即删去了blob.num 项。

        通过调用此API进行处理，具体python函数如下：

        convert\_bproto\_to\_npy.py

**[python]** [view plaincopy](http://blog.csdn.net/deeplearninglc007/article/details/40086503)

1. #!/usr/bin/python
3. **import** numpy as np
4. **from** caffe.io **import** blobproto\_to\_array2
5. **from** caffe.proto **import** caffe\_pb2
7. blob = caffe\_pb2.BlobProto()
9. filename = './imagenet\_mean.binaryproto'
11. data = open(filename, "rb").read()
12. blob.ParseFromString(data)
14. nparray =blobproto\_to\_array2(blob)
15. f = file("mean.npy","wb")
16. np.save(f,nparray)
17. f.close()

        c.      按照demo步骤进行分类：

        代码的主要部分参照

        classify.py

* <http://nbviewer.ipython.org/github/BVLC/caffe/blob/master/examples/imagenet_classification.ipynb>

        classifymap.py

* <http://nbviewer.ipython.org/github/BVLC/caffe/blob/master/examples/net_surgery.ipynb>

        输出部分的说明

        1)     Classify.py

        输出的是prediction[0],一维数组，以imagenet为例就是大小为1000的数组，每个值对应概率大小。

        处理部分代码：

**[python]** [view plaincopy](http://blog.csdn.net/deeplearninglc007/article/details/40086503)

1. preDict = {}
2. **for** i **in** xrange(len(prediction[0])):
3. preDict[prediction[0][i]] = i
4. **for** i **in** xrange(5):
5. val = sorted(preDict.keys())[-i -1]
6. **print**("%d %f %s" %(preDict[val], val \* 100, linecache.getline(SYNSET, preDict[val])))

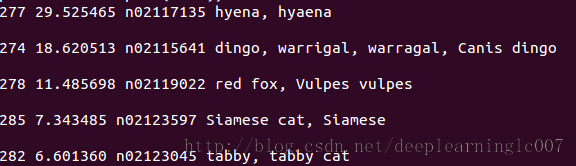
        把数组的值和索引对应到字典中，值作为键，把数组值进行排序，再用前五个作为键，找出索引值，也就是对应的类别。

为了能够直观地显示数字所代表的类别名，需要输出synset\_words.txt文件中对应行数的内容。这里用的是linecache.getline()函数，需要

        import caffe

        为了修改方便，把synset\_words.txt的路径设为变量

        SYNSET = '../../data/ilsvrc12/synset\_words.txt'



        （序号概率值（百分比）对应种类）

        2)     Classifymap.py

        输出的是

                     outMat =out['prob'][0].argmax(axis=0)

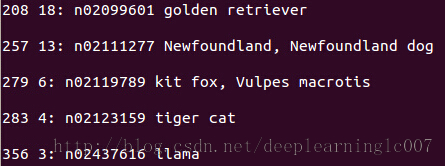
是一个二维矩阵，8\*8大小，每个值对应的都是一种类别，出现的越多代表概率越高。

        处理部分代码：

**[plain]** [view plaincopy](http://blog.csdn.net/deeplearninglc007/article/details/40086503)

1. h1, w1 = np.shape(outMat)
2. outList = list(np.reshape(outMat,h1 \* w1))
3. #print(outList)
5. outSet = set(outList)
6. outdict = {}
7. for x in outSet:
8. outdict[outList.count(x)]=x
10. appear = outdict.keys()
11. applen = len(appear)
12. if len <= 5:
13. for i in xrange(applen):
14. print('%d %d: %s'%(outdict[appear[-(i+1)]], appear[-(i+1)],linecache.getline(SYNSET,outdict[appear[-(i+1)]])))
15. else:
16. for i in xrange(5):
17. print('%d %d: %s'%(outdict[appear[-(i+1)]], appear[-(i+1)], linecache.getline(SYNSET,outdict[appear[-(i+1)]])))

        和上面的文件大致思路相同。但是需要先把矩阵展开成一维列表，用set()变成集合，也就是去掉重复元素，再一一对应到字典中，然后通过count()找到每个值在矩阵中出现的次数，进行排序即可。



        （序号出现次数对应种类）

        3)  源代码：

        Classiy.py

**[python]** [view plaincopy](http://blog.csdn.net/deeplearninglc007/article/details/40086503)

1. **import** numpy as np
2. **import** matplotlib.pyplot as plt
3. **import** pylab
5. **import** caffe
7. caffe\_root = '../'
9. MODEL\_FILE = '../imagenet/imagenet\_deploy.prototxt'
10. PRETRAINED = '../imagenet/caffe\_reference\_imagenet\_model'
11. IMAGE\_FILE = '../images/cat.jpg'
13. #net = caffe.Classifier(MODEL\_FILE, PRETRAINED, mean\_file = caffe\_root + '../python/caffe/imagenet/ilsvrc\_2012\_mean.npy', channel\_swap = (2,1,0), input\_scale = 255)
15. net = caffe.Classifier(MODEL\_FILE, PRETRAINED, mean\_file = caffe\_root + 'mean2.npy', channel\_swap = (2,1,0), input\_scale = 255)
17. net.set\_phase\_test()
18. net.set\_mode\_gpu()
20. input\_image = caffe.io.load\_image(IMAGE\_FILE)
21. pylab.ion()
22. plt.imshow(input\_image)
23. #pylab.show()

26. prediction = net.predict([input\_image])
27. **print** 'prediction shape:', prediction[0].shape
28. #print(prediction[0])
29. plt.plot(prediction[0])
30. #pylab.show()
31. preDict = {}
32. preList = list(prediction[0])
33. **for** i **in** preList:
34. preDict[preList[i]] = i
35. preLen = len(preList)
36. **for** i **in** xrange(5):
37. **print**('%d %d: %s' %(preDict[preDict.keys[-(i+1)]], preDict.values[-(i+1)], linecache.getline(SYNSET,preDict.values[-(i+1)])))

classifymap.py

**[python]** [view plaincopy](http://blog.csdn.net/deeplearninglc007/article/details/40086503)

1. **import** caffe
2. **import** matplotlib.pyplot as plt
3. **import** pylab
4. **import** numpy as np
5. **import** linecache
7. IMAGE\_FILE ='../images/dog.jpg'
8. SYNSET = '../../data/ilsvrc12/synset\_words.txt'
9. # Load the original network and extract the fully-connected layers' parameters.
11. net = caffe.Net('../imagenet/imagenet\_deploy.prototxt', '../imagenet/caffe\_reference\_imagenet\_model')
13. params = ['fc6', 'fc7', 'fc8']
15. # fc\_params = {name: (weights, biases)}
17. fc\_params = {pr: (net.params[pr][0].data, net.params[pr][1].data) **for** pr **in** params}


21. #for fc in params:
23. #print '{} weights are {} dimensional and biases are {} dimensional'.format(fc, fc\_params[fc][0].shape, fc\_params[fc][1].shape)
25. # Load the fully-convolutional network to transplant the parameters.
27. net\_full\_conv = caffe.Net('../imagenet/imagenet\_full\_conv.prototxt', '../imagenet/caffe\_reference\_imagenet\_model')
29. params\_full\_conv = ['fc6-conv', 'fc7-conv', 'fc8-conv']
31. # conv\_params = {name: (weights, biases)}
33. conv\_params = {pr: (net\_full\_conv.params[pr][0].data, net\_full\_conv.params[pr][1].data) **for** pr **in** params\_full\_conv}


37. #for conv in params\_full\_conv:
39. #print '{} weights are {} dimensional and biases are {} dimensional'.format(conv, conv\_params[conv][0].shape, conv\_params[conv][1].shape)
41. **for** pr, pr\_conv **in** zip(params, params\_full\_conv):
43. conv\_params[pr\_conv][1][...] = fc\_params[pr][1]
44. **for** pr, pr\_conv **in** zip(params, params\_full\_conv):
46. out, in\_, h, w = conv\_params[pr\_conv][0].shape
48. W = fc\_params[pr][0].reshape((out, in\_, h, w))
50. conv\_params[pr\_conv][0][...] = W
52. # net\_full\_conv.save('../imagenet/caffe\_imagenet\_full\_conv')
53. # load input and configure preprocessing
55. im = caffe.io.load\_image(IMAGE\_FILE)
57. plt.imshow(im)
58. #pylab.show()
60. net\_full\_conv.set\_mean('data', '../../python/caffe/imagenet/ilsvrc\_2012\_mean.npy')
62. net\_full\_conv.set\_channel\_swap('data', (2,1,0))
64. net\_full\_conv.set\_input\_scale('data', 255.0)
66. # make classification map by forward pass and show top prediction index per location
68. out = net\_full\_conv.forward\_all(data=np.asarray([net\_full\_conv.preprocess('data', im)]))
70. outMat = out['prob'][0].argmax(axis=0)
72. h1, w1 = np.shape(outMat)
73. outList = list(np.reshape(outMat,h1 \* w1))
74. #print(outList)
76. outSet = set(outList)
77. outdict = {}
78. **for** x **in** outSet:
79. outdict[outList.count(x)]=x
81. appear = outdict.keys()
82. applen = len(appear)
83. **if** len <= 5:
84. **for** i **in** xrange(applen):
85. **print**('%d %d: %s' %(outdict[appear[-(i+1)]], appear[-(i+1)], linecache.getline(SYNSET,outdict[appear[-(i+1)]])))
86. **else**:
87. **for** i **in** xrange(5):
88. **print**('%d %d: %s' %(outdict[appear[-(i+1)]], appear[-(i+1)], linecache.getline(SYNSET,outdict[appear[-(i+1)]])))

在训练之前，要准备训练需要的训练集，为了验证训练的效果，最好也准备一定数量的测试集。由于caffe的数据集的输入是leveldb格式，因此在训练前需要将输入转为相应格式。

在caffe\_root/example 文件夹中，提供了一些例子，cifar10与imagenet均是将图片形式数据库转换为leveldb格式，mnist则是将其本身的数据集转换为leveldb格式。这就要求我们需要把自己的数据集转换成leveldb格式，需要自己编写程序实现。下面以jpg格式图片为例说明。

在此，假设数据库本身是以图片形式给出，并且给出了label或者同一类别的图片已经分类。这样我们就可以通过imagenet例子中给出的create\_imagenet.sh 对我们自己的数据库进行转换，要求数据集图片格式为jpg，以下为具体操作步骤：

A．若数据集已经给出label则忽略此步骤。图片按照类别装在不同文件夹下的情况，自己编写mklabel.sh命令，对图片进行处理并标明label。具体操作参照mklabel.sh 程序说明。

**[cpp]** [view plaincopy](http://blog.csdn.net/deeplearninglc007/article/details/40086503)

1. <pre name="code" **class**="plain">mklabel.sh
2. #!/bin/sh
4. #----------------------------------------------------
5. #文件存放形式为
6. #   dir/subdir1/files...
7. #   dir/subdir2/files...
8. #   dir/subdir3/files...
9. #   dir/subdirX/files...
11. #用法：
12. #1.$ sh mklabel.sh dir startlabel ;dir 为目标文件夹名称
13. #2.$ chmod a+x mklabel.sh ；然后可以直接用文件名运行
14. #3.默认label信息显示在终端，请使用转向符'>'生成文本，例：
15. #       $ sh ./mklabel.sh  data/faces94/male  > label.txt
16. #4.确保文件夹下除了图片不含其他文件(若含有则需自行添加判断语句)
17. #-----------------------------------------------------
19. DIR=~/codes/mklabel.sh      #命令位置（无用）
20. label=1                 #label起始编号(为数字，根据自己需要修改)
21. testnum=0               #保留的测试集大小
23. **if** test $# -eq 0;then   #无参数，默认为当前文件夹下，label=1
24. $DIR . 0 $label
25. **else**
26. **if** test $# -eq 1;then   #仅有位置参数，默认testnum=0,label=1
27. $DIR $1 0 $label
28. **else**
29. **if** test $# -eq 2;then   #两个参数时,label=1
30. $DIR $1 $2 $label
31. **else**
32. testnum=$2          #每个类别保留测试集大小
33. label=$3            #自定义label起始
35. cd $1               #转到目标文件夹
37. **if** test $testnum -ne 0;then
38. mkdir "testdata"    #建立测试集
39. fi
41. **for** i in \* ; **do**
42. exist=`expr "$i" != "testdata"`
43. **if** test -d $i && test $exist -eq 1;then #文件夹存在
44. #echo
45. #echo 'DIR:' $i
47. cd $i           #进入文件夹
48. num=1       #图片数目
49. **for** j in \*
50. **do**
51. **if** test $num -gt $testnum;then
52. echo  $j  $label
53. mv $j ../
54. fi
55. num=`expr $num + 1`
56. done
57. cd ..           #回到上层目录
59. **if** test $testnum -eq 0;then
60. rmdir $i
61. **else**
62. mv $i ./testdata
63. fi
65. label=`expr $label + 1`
66. #计算label
67. fi
68. done
69. fi
70. fi
71. fi

B．修改create\_imagenet.sh 文件，使其中的图片源地址与标明label的txt文件对应到自己数据库的相应文件。其中的convert\_imageset.cpp 函数的参数依次为图片文件夹位置，label.txt文本文件，目标数据集名称，是否随机存储图片与label（即打乱图片数据的读入顺序）。

若你所使用的数据集不是通过图片形式给出，为了方便，可以根据leveldb数据的存储格式，自己编写程序转换数据集。

C．多通道图片需要用到meanfile，通过example/imagenet文件夹下的shell函数make\_imagenet\_mean.sh，更改相应函数，很轻松的得到binaryproto文件，在训练时可能需要用到。

**step 3.**  使用自己的数据进行训练：

以最简单的mnist网络为例，因为数据集的不同则需要更改的参数也不同。在训练前最好在example下新建个文件夹命名为数据集的名称，然后把mnist下的5个文件：

        lenet.prototxt

        lenet\_solver.prototxt

        lenet\_train.prototxt

        lenet\_test.prototxt

        train\_lenet.sh

      复制到新文件夹下，以上5个文件为必需的文件。按顺序对以上文件进行修改，在忽略网络结构的情况下，需要修改的有:

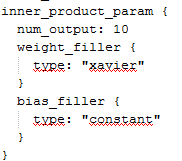
a.      lenet.prototxt：

**[plain]** [view plaincopy](http://blog.csdn.net/deeplearninglc007/article/details/40086503)

1. input\_dim: 64
2. input\_dim: 1
3. input\_dim: 28
4. input\_dim: 28

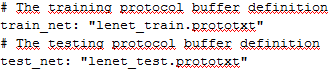
分别为一次导入的图片个数，channel，heigth ，width。

                      倒数第二层，即输入给softmax层数据的那层，其中的num\_output需要更改为实际图片的label数，即图片类别数。否则在进行训练时，会出现错误。



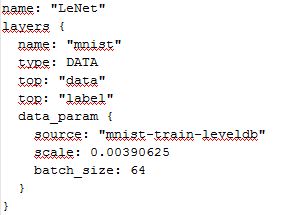
b.      lenet\_solver.prototxt：

       如果之前文件名没改的话则不需要更改以上两项，否则改为对应文件。其他参数根据实际需要更改。

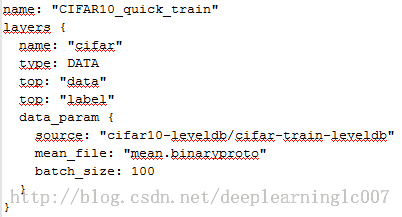


c.      lenet\_train.prototxt：

        需要把data层的数据源文件替换为自己的数据。



        在训练多通道图片时，此处最好需要有一个meanfile参数。例如cifar10



       num\_output参数参照lenet.prototxt修改。

d.      lenet\_test.prototxt：

        参照lenet\_train.prototxt 进行相应修改。

e.      train\_lenet.sh：

         lenet\_solver.prototxt文件名未更改的话则不需要进行改动。

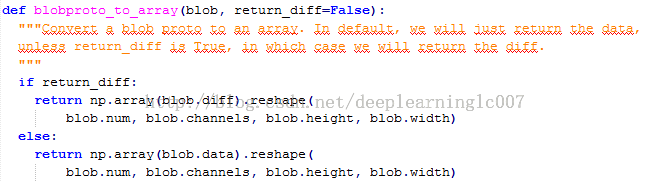
**step 4.**  使用自己的model进行分类：

        Reference: <http://nbviewer.ipython.org/github/BVLC/caffe/blob/master/examples/classification.ipynb>

         假设之前的所有步骤已经成功，并且已经训练出正确率还行的model，则现在就可以使用model对图片进行分类。

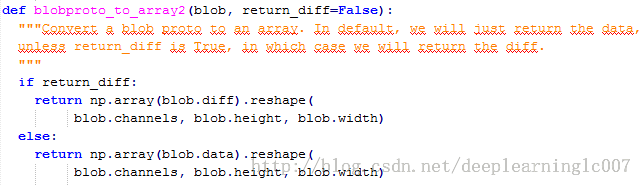
        a.      首先确保已经正确安装了pythonwrapper，以及caffe\_root/python/文件夹下requirements.txt文件中所需要的组件。

        b.      另外，还需要meanfile的npy文件。即需要把数据对应的binaryproto文件转换到npy文件。Caffe\_root/python/caffe/io.cpp文件中已经给了对应的API。

****

        具体参照：<https://github.com/BVLC/caffe/issues/420>

        需要对blobproto\_to\_array 进行修改，得到blobproto\_to\_array2.



        即删去了blob.num 项。

        通过调用此API进行处理，具体python函数如下：

        convert\_bproto\_to\_npy.py

**[python]** [view plaincopy](http://blog.csdn.net/deeplearninglc007/article/details/40086503)

1. #!/usr/bin/python
3. **import** numpy as np
4. **from** caffe.io **import** blobproto\_to\_array2
5. **from** caffe.proto **import** caffe\_pb2
7. blob = caffe\_pb2.BlobProto()
9. filename = './imagenet\_mean.binaryproto'
11. data = open(filename, "rb").read()
12. blob.ParseFromString(data)
14. nparray =blobproto\_to\_array2(blob)
15. f = file("mean.npy","wb")
16. np.save(f,nparray)
17. f.close()

        c.      按照demo步骤进行分类：

        代码的主要部分参照

        classify.py

* <http://nbviewer.ipython.org/github/BVLC/caffe/blob/master/examples/imagenet_classification.ipynb>

        classifymap.py

* <http://nbviewer.ipython.org/github/BVLC/caffe/blob/master/examples/net_surgery.ipynb>

        输出部分的说明

        1)     Classify.py

        输出的是prediction[0],一维数组，以imagenet为例就是大小为1000的数组，每个值对应概率大小。

        处理部分代码：

**[python]** [view plaincopy](http://blog.csdn.net/deeplearninglc007/article/details/40086503)

1. preDict = {}
2. **for** i **in** xrange(len(prediction[0])):
3. preDict[prediction[0][i]] = i
4. **for** i **in** xrange(5):
5. val = sorted(preDict.keys())[-i -1]
6. **print**("%d %f %s" %(preDict[val], val \* 100, linecache.getline(SYNSET, preDict[val])))

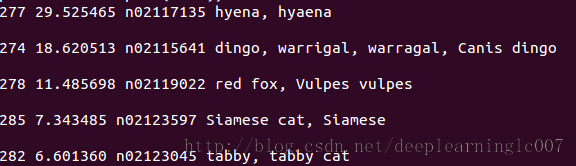
        把数组的值和索引对应到字典中，值作为键，把数组值进行排序，再用前五个作为键，找出索引值，也就是对应的类别。

为了能够直观地显示数字所代表的类别名，需要输出synset\_words.txt文件中对应行数的内容。这里用的是linecache.getline()函数，需要

        import caffe

        为了修改方便，把synset\_words.txt的路径设为变量

        SYNSET = '../../data/ilsvrc12/synset\_words.txt'



        （序号概率值（百分比）对应种类）

        2)     Classifymap.py

        输出的是

                     outMat =out['prob'][0].argmax(axis=0)

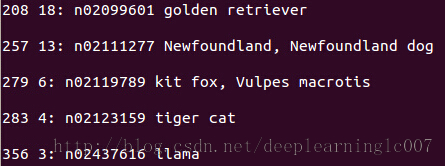
是一个二维矩阵，8\*8大小，每个值对应的都是一种类别，出现的越多代表概率越高。

        处理部分代码：

**[plain]** [view plaincopy](http://blog.csdn.net/deeplearninglc007/article/details/40086503)

1. h1, w1 = np.shape(outMat)
2. outList = list(np.reshape(outMat,h1 \* w1))
3. #print(outList)
5. outSet = set(outList)
6. outdict = {}
7. for x in outSet:
8. outdict[outList.count(x)]=x
10. appear = outdict.keys()
11. applen = len(appear)
12. if len <= 5:
13. for i in xrange(applen):
14. print('%d %d: %s'%(outdict[appear[-(i+1)]], appear[-(i+1)],linecache.getline(SYNSET,outdict[appear[-(i+1)]])))
15. else:
16. for i in xrange(5):
17. print('%d %d: %s'%(outdict[appear[-(i+1)]], appear[-(i+1)], linecache.getline(SYNSET,outdict[appear[-(i+1)]])))

        和上面的文件大致思路相同。但是需要先把矩阵展开成一维列表，用set()变成集合，也就是去掉重复元素，再一一对应到字典中，然后通过count()找到每个值在矩阵中出现的次数，进行排序即可。



（序号出现次数对应种类）

3)  源代码：

Classiy.py

**[python]** [view plaincopy](http://blog.csdn.net/deeplearninglc007/article/details/40086503)

1. **import** numpy as np
2. **import** matplotlib.pyplot as plt
3. **import** pylab
5. **import** caffe
7. caffe\_root = '../'
9. MODEL\_FILE = '../imagenet/imagenet\_deploy.prototxt'
10. PRETRAINED = '../imagenet/caffe\_reference\_imagenet\_model'
11. IMAGE\_FILE = '../images/cat.jpg'
13. #net = caffe.Classifier(MODEL\_FILE, PRETRAINED, mean\_file = caffe\_root + '../python/caffe/imagenet/ilsvrc\_2012\_mean.npy', channel\_swap = (2,1,0), input\_scale = 255)
15. net = caffe.Classifier(MODEL\_FILE, PRETRAINED, mean\_file = caffe\_root + 'mean2.npy', channel\_swap = (2,1,0), input\_scale = 255)
17. net.set\_phase\_test()
18. net.set\_mode\_gpu()
20. input\_image = caffe.io.load\_image(IMAGE\_FILE)
21. pylab.ion()
22. plt.imshow(input\_image)
23. #pylab.show()

26. prediction = net.predict([input\_image])
27. **print** 'prediction shape:', prediction[0].shape
28. #print(prediction[0])
29. plt.plot(prediction[0])
30. #pylab.show()
31. preDict = {}
32. preList = list(prediction[0])
33. **for** i **in** preList:
34. preDict[preList[i]] = i
35. preLen = len(preList)
36. **for** i **in** xrange(5):
37. **print**('%d %d: %s' %(preDict[preDict.keys[-(i+1)]], preDict.values[-(i+1)], linecache.getline(SYNSET,preDict.values[-(i+1)])))

classifymap.py

**[python]** [view plaincopy](http://blog.csdn.net/deeplearninglc007/article/details/40086503)

1. **import** caffe
2. **import** matplotlib.pyplot as plt
3. **import** pylab
4. **import** numpy as np
5. **import** linecache
7. IMAGE\_FILE ='../images/dog.jpg'
8. SYNSET = '../../data/ilsvrc12/synset\_words.txt'
9. # Load the original network and extract the fully-connected layers' parameters.
11. net = caffe.Net('../imagenet/imagenet\_deploy.prototxt', '../imagenet/caffe\_reference\_imagenet\_model')
13. params = ['fc6', 'fc7', 'fc8']
15. # fc\_params = {name: (weights, biases)}
17. fc\_params = {pr: (net.params[pr][0].data, net.params[pr][1].data) **for** pr **in** params}


21. #for fc in params:
23. #print '{} weights are {} dimensional and biases are {} dimensional'.format(fc, fc\_params[fc][0].shape, fc\_params[fc][1].shape)
25. # Load the fully-convolutional network to transplant the parameters.
27. net\_full\_conv = caffe.Net('../imagenet/imagenet\_full\_conv.prototxt', '../imagenet/caffe\_reference\_imagenet\_model')
29. params\_full\_conv = ['fc6-conv', 'fc7-conv', 'fc8-conv']
31. # conv\_params = {name: (weights, biases)}
33. conv\_params = {pr: (net\_full\_conv.params[pr][0].data, net\_full\_conv.params[pr][1].data) **for** pr **in** params\_full\_conv}


37. #for conv in params\_full\_conv:
39. #print '{} weights are {} dimensional and biases are {} dimensional'.format(conv, conv\_params[conv][0].shape, conv\_params[conv][1].shape)
41. **for** pr, pr\_conv **in** zip(params, params\_full\_conv):
43. conv\_params[pr\_conv][1][...] = fc\_params[pr][1]
44. **for** pr, pr\_conv **in** zip(params, params\_full\_conv):
46. out, in\_, h, w = conv\_params[pr\_conv][0].shape
48. W = fc\_params[pr][0].reshape((out, in\_, h, w))
50. conv\_params[pr\_conv][0][...] = W
52. # net\_full\_conv.save('../imagenet/caffe\_imagenet\_full\_conv')
53. # load input and configure preprocessing
55. im = caffe.io.load\_image(IMAGE\_FILE)
57. plt.imshow(im)
58. #pylab.show()
60. net\_full\_conv.set\_mean('data', '../../python/caffe/imagenet/ilsvrc\_2012\_mean.npy')
62. net\_full\_conv.set\_channel\_swap('data', (2,1,0))
64. net\_full\_conv.set\_input\_scale('data', 255.0)
66. # make classification map by forward pass and show top prediction index per location
68. out = net\_full\_conv.forward\_all(data=np.asarray([net\_full\_conv.preprocess('data', im)]))
70. outMat = out['prob'][0].argmax(axis=0)
72. h1, w1 = np.shape(outMat)
73. outList = list(np.reshape(outMat,h1 \* w1))
74. #print(outList)
76. outSet = set(outList)
77. outdict = {}
78. **for** x **in** outSet:
79. outdict[outList.count(x)]=x
81. appear = outdict.keys()
82. applen = len(appear)
83. **if** len <= 5:
84. **for** i **in** xrange(applen):
85. **print**('%d %d: %s' %(outdict[appear[-(i+1)]], appear[-(i+1)], linecache.getline(SYNSET,outdict[appear[-(i+1)]])))
86. **else**:
87. **for** i **in** xrange(5):
88. **print**('%d %d: %s' %(outdict[appear[-(i+1)]], appear[-(i+1)], linecache.getline(SYNSET,outdict[appear[-(i+1)]])))