# AI培训作业

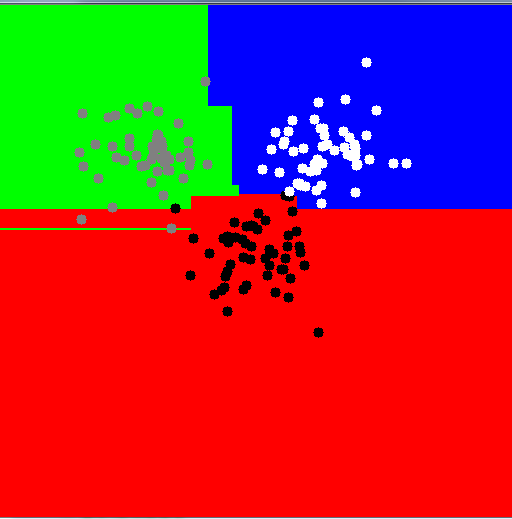
## 1.使用【随机森林】、【AdaBoost】、【MLP】三种模型进行训练和预测

【随机森林】

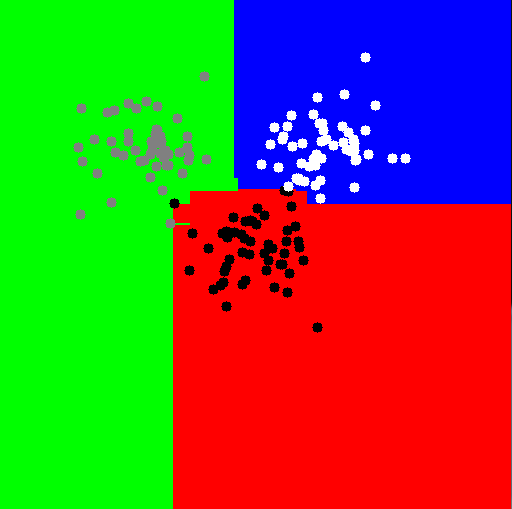
由于代码都是在书中是现成的。

主要调节的参数有树的数量num以及树的深度的depth。但测试中发现对结果影响最大的参数是 树的数量num 下面给出不同树的数量的结果（树的深度固定为5）：

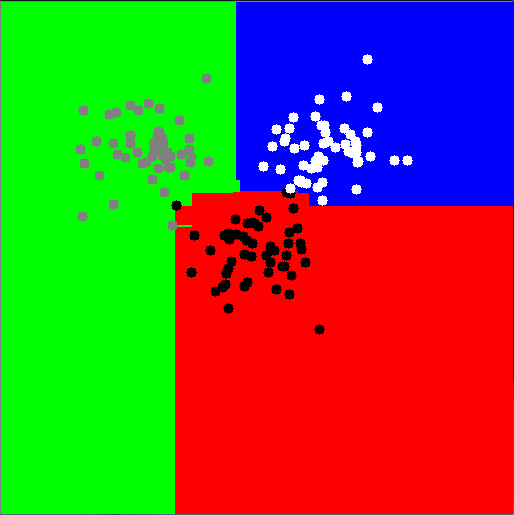
（1）num 4



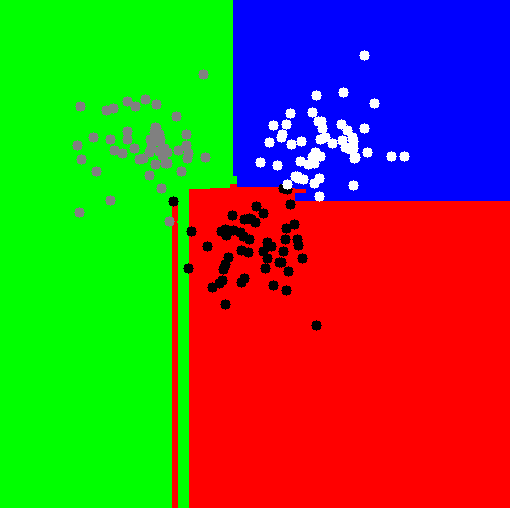
1. num 5



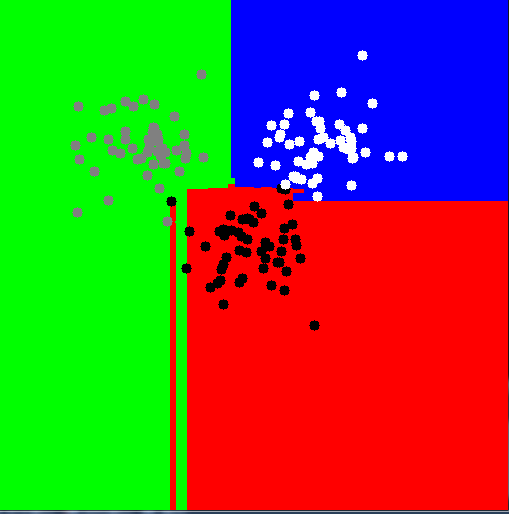
（3）num 6



（4）num 7

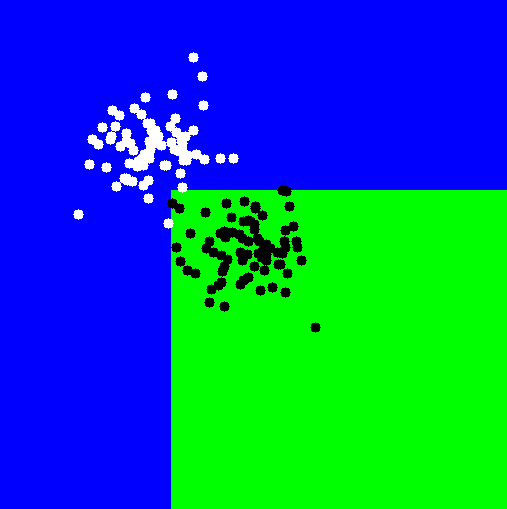


（5）num 8



【AdaBoost】

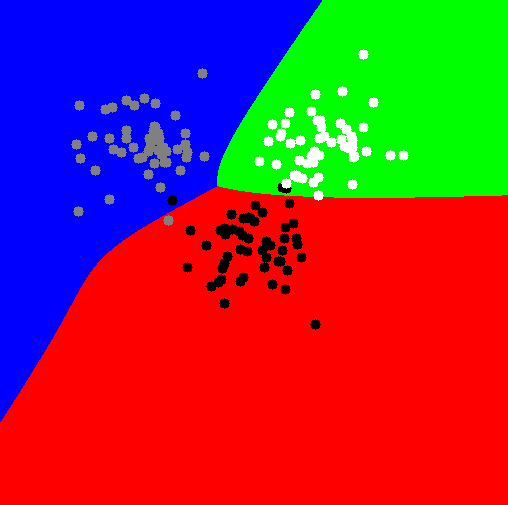
虽然adaboost模型提供了弱分类器个数、权重训练率、最大深度等参数，但由于二维数据的二分类问题比较简单，所以调节这些参数，得到的分类结果都一样。



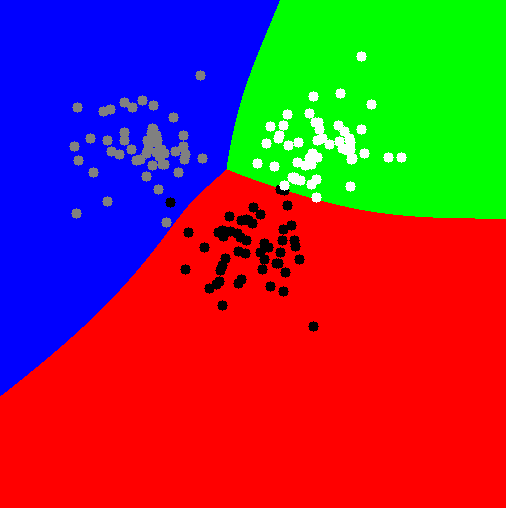
【MLP】

神经网络用来进行多分类的问题训练和测试，可以调节的参数比较多，包括网络结构（层数、隐含层节点数等）、学习率、动量系数等。下面举一些不同网络结构的测试例子：

三层网络：2,10,3



四层网络：2,6,5,3

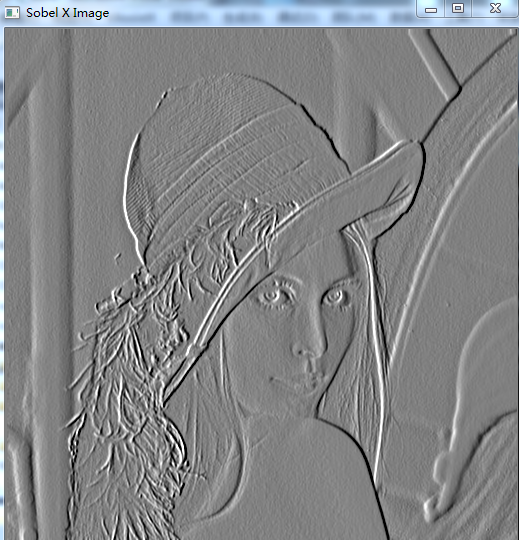


## 2.边缘检测、图像旋转与缩放、直线检测

【sobel算子、canny算子】

首先利用不同sobel算子对lena图像进行各种变换：

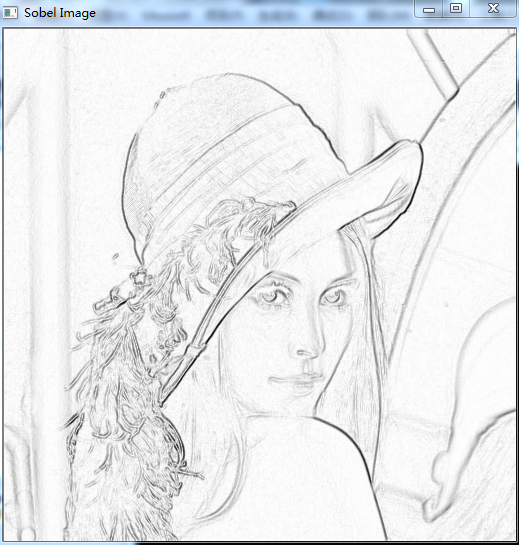
X方向sobel算子：



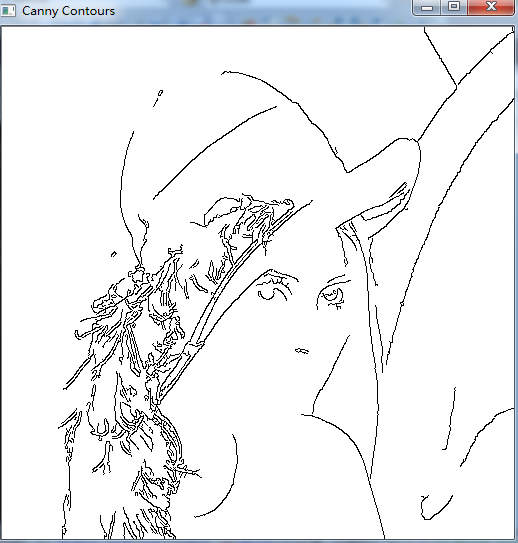
Y方向sobel算子：



X+Y方向sobel算子模：

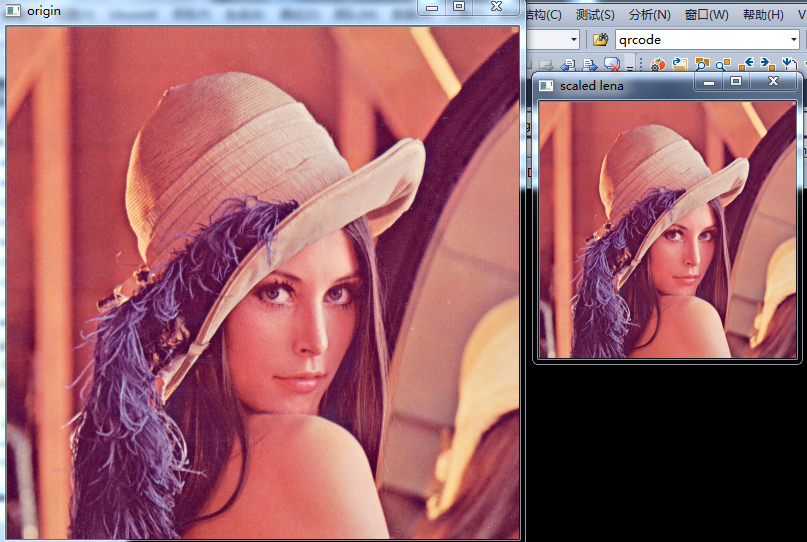


利用canny算子检测后生成的反转图像：

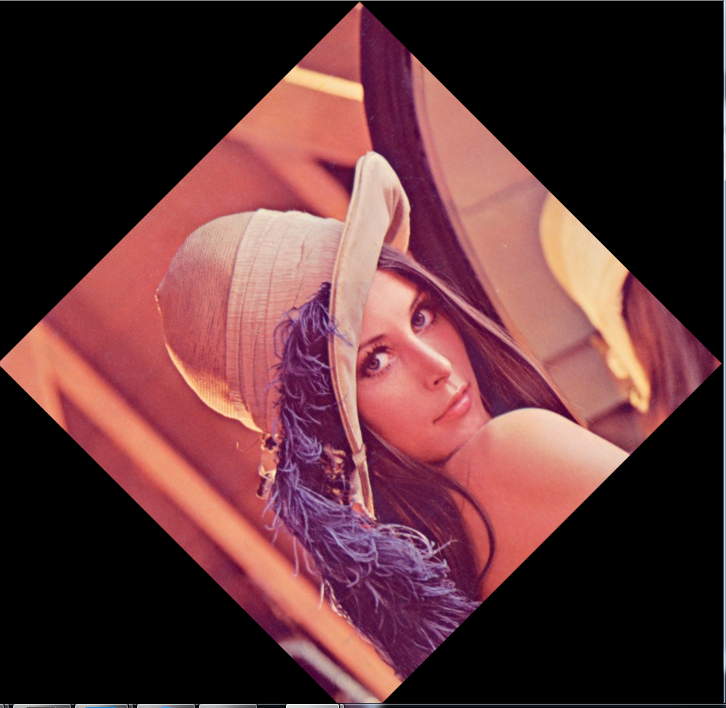


【图像旋转与缩放】

图像缩小为原图的一半：

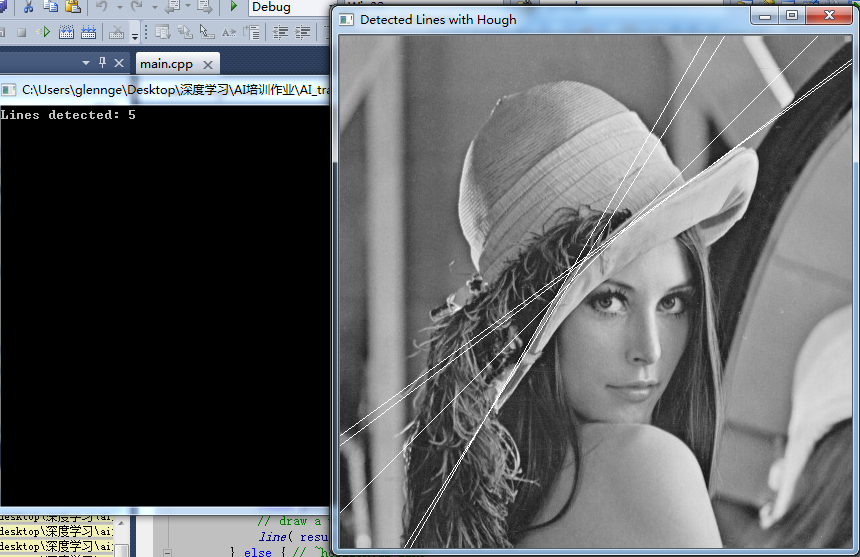


图像逆时针旋转45度：



【哈夫变换直线检测】

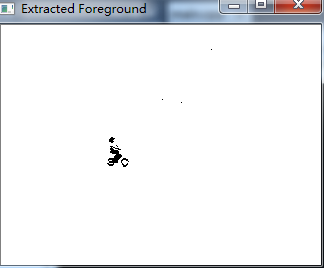
测试中哈夫变换的也可以调节不同的参数，对应检测的直线数量也会有所不同，下面结果使用的参数是rho 1， theta为1度，threshold为120：



## 3.背景建模、运动检测

【高斯混合模型】

Opencv中有一个cv::BackgroundSubtractorMOG的类用来做高斯混合模型，检测结果如下：



【VIBE背景建模】

根据VIBE官方提供的opencv源码，进行VIBE算法测试，结果如下：

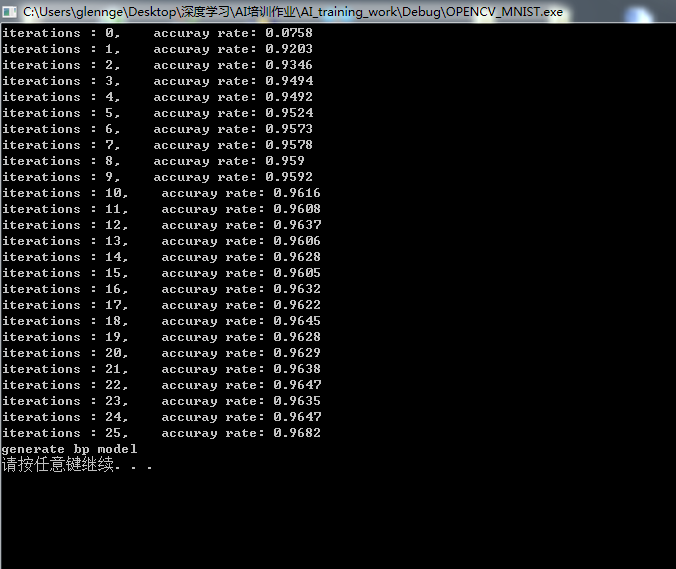


## 4.opencv多层前馈神经网络MNIST数据集训练与测试

对于MNIST前馈神经网络的训练这里尝试了两种源码测试，一种为网上的直接写的三层前馈神经网络的源码，进行训练，其中输入层为28\*28=784维，输出层为0-1编码，故有10维。

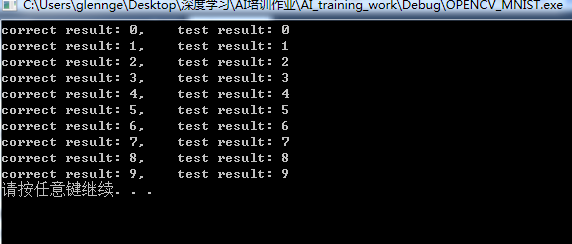
4.1 三层前馈神经网络源码训练和测试：





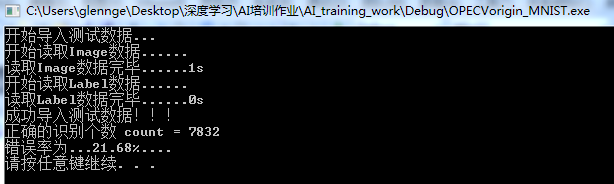
最后随机选择十张数字图像进行测试，结果如下：

0 1 2 3 4 5 6 7 8 9

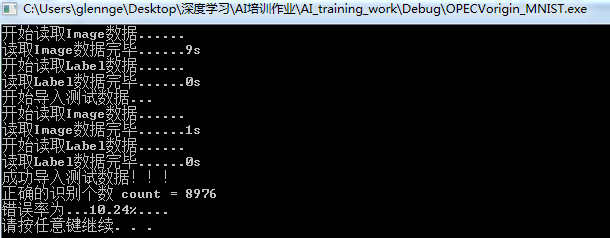


4.2 基于opencv前馈神经网络训练和测试

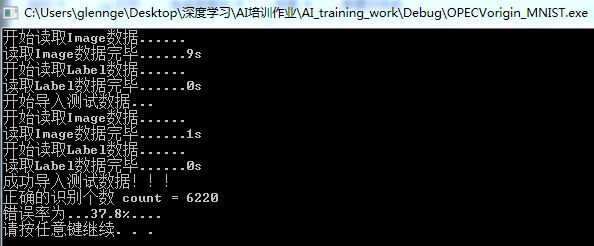
迭代次数 5 权重学习率 0.01 隐含层神经元200 测试结果



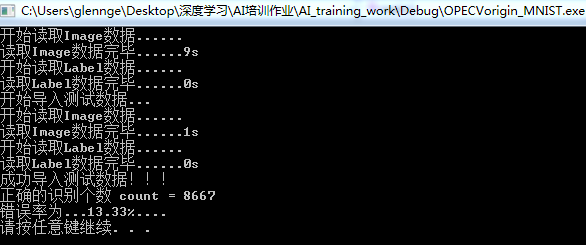
迭代次数 5 权重学习率 0.1 隐含层神经元200 测试结果



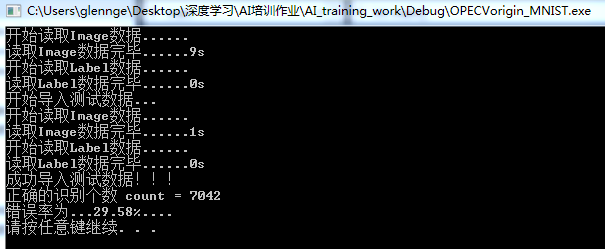
迭代次数 5 权重学习率 0.5 隐含层神经元200 测试结果



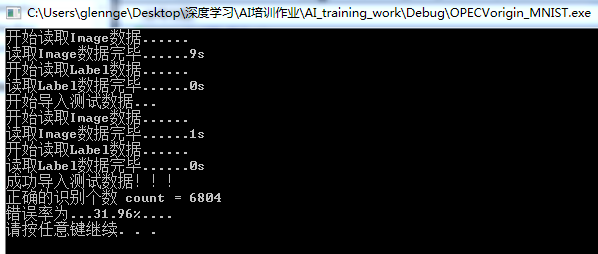
迭代次数 10 权重学习率 0.1 隐含层神经元200 测试结果



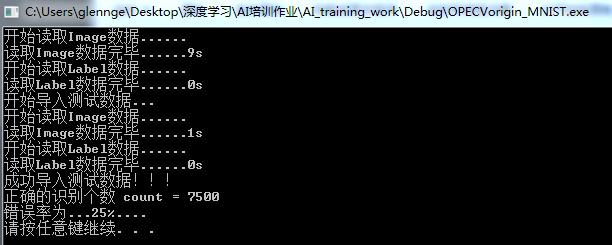
迭代次数 5权重学习率 0.1 隐含层神经元100测试结果



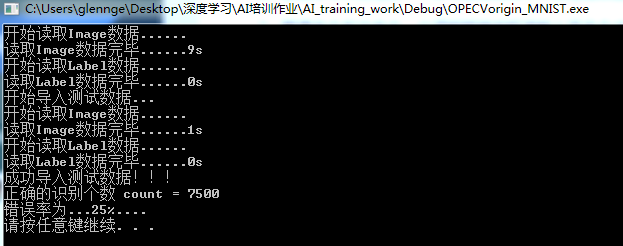
迭代次数10权重学习率 0.1 隐含层神经元100测试结果



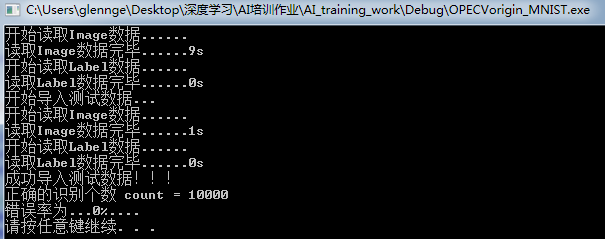
迭代次数 10 权重学习率 0.1 隐含层神经元50\*50测试结果



迭代次数 10 权重学习率 0.1 隐含层神经元100\*100测试结果



迭代次数 10 权重学习率 0.25 隐含层神经元100\*100测试结果

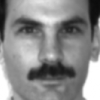


最后竟然跑出来一个准确率为100%！！的模型

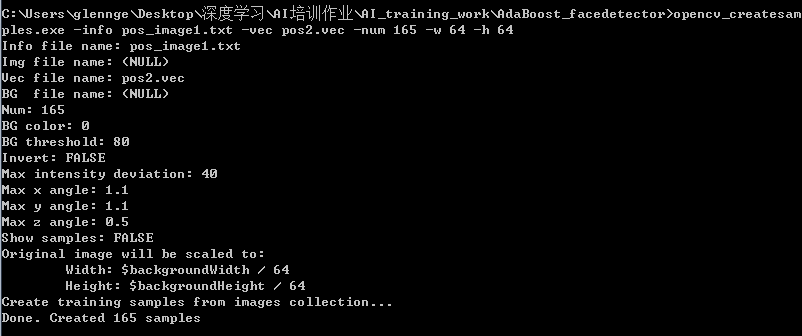
只能说前馈神经网络是调参的玄学。。。。

## 5.opencv Adaboost人脸检测模型训练与测试

以yale database中的165张人脸作为正样本，以Weizmann\_Seg\_DB中的200张图像作为负样本。进行adaboost人脸检测模型训练。

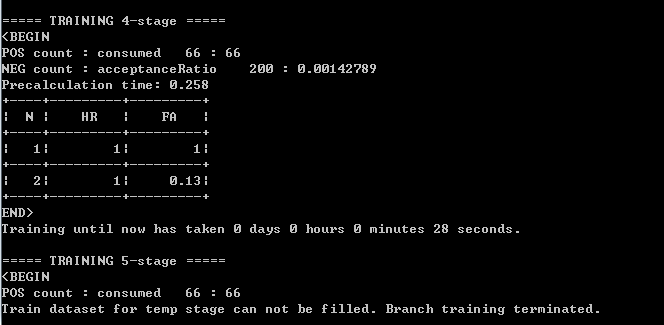
使用opencv\_createsample.exe生成正样本的vec文件，大小归一化为64\*64：



再使用opencv\_traincascade.exe进行训练，训练中正负样本分别为66:200近似1:3，使用HOG特征进行训练，adaboost级数不超过10级。

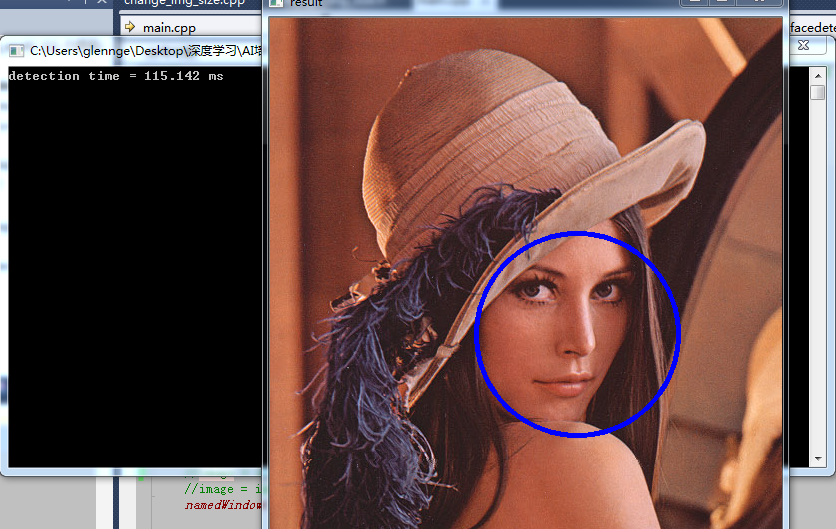


由于正负样本数量都比较少，因此训练的速度较快，层数也比较小。



测试结果如下，虽然测试结果在经典图像lena准确，但是该模型泛化能力和测试效果在其他复杂图片中都不是很好。如果要提高，需要更多的人脸样本和负样本进行训练。

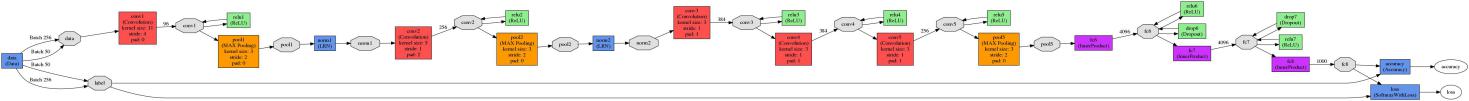




## 6.基于imagenet的AlexNET图像分类模型

为了更好了解alexnet的结构，利用caffe自带的python脚本可视化网络结果：

python ./python/draw\_net.py models/bvlc\_reference\_caffenet/train\_val.prototxt alexnet.jpg

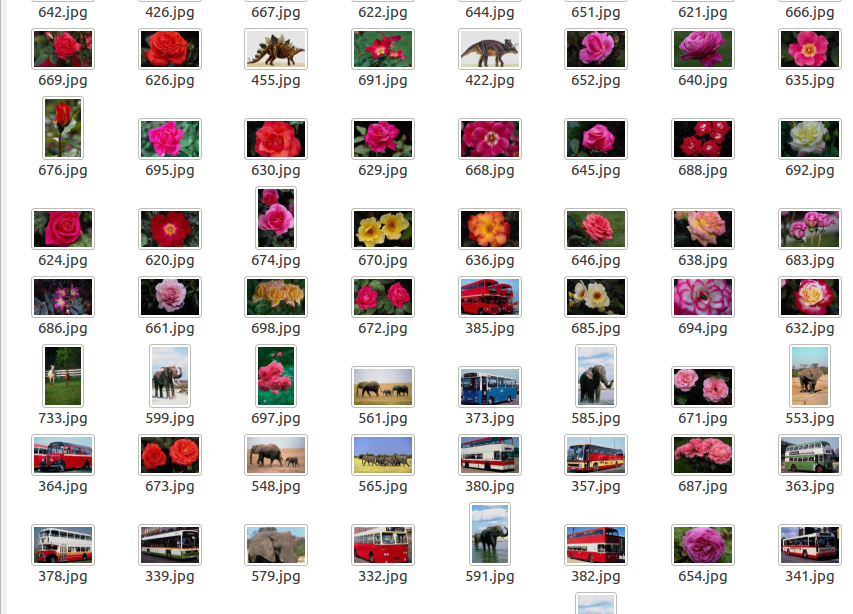


6.1 基于Alexnet网络结构，利用自己的数据集训练分类模型并测试

（1）准备数据

虽然imagenet官网上有数据图片集，由于数据量太大，单次训练也是一件漫长的过程，为了熟悉整个流程并且调整参数测试，因此这里找了网上一个人的数据集来代替http://www.cnblogs.com/denny402/p/5083300.html。

该数据集有500张图片，分为大巴车、恐龙、大象、鲜花和马五个类，每个类100张。编号分别以3，4，5，6，7开头，各为一类。我从其中每类选出20张作为测试，其余80张作为训练。因此最终训练图片400张，测试图片100张，共5。



将上述图片放在我自己创建了一个文件目录下面训练文件glenn/mycaffe /myimagenet/train/和测试文件glenn/mycaffe/myimagenet/test/下面。

（2） 数据处理转换

首先在myimagenet文件夹下创建一个myconfig文件，用来存放配置文件和脚本文件，，然后编写一个shell脚本create\_filelist.sh，用来生成train.txt和test.txt清单文件：

…/myconfig$sudo vi create\_filelist.sh

编辑文件，内容为，然后保存（注意文本中的路径）

#!/usr/bin/env sh

DATA=mycaffe/myimagenet/

MY=mycaffe/myimagenet/myconfig

echo "Create train.txt..."

rm -rf $MY/train.txt

for i in 3 4 5 6 7

do

find $DATA/train -name $i\*.jpg | cut -d '/' -f4-5 | sed "s/$/ $i/">>$MY/train.txt

done

echo "Create test.txt..."

rm -rf $MY/test.txt

for i in 3 4 5 6 7

do

find $DATA/test -name $i\*.jpg | cut -d '/' -f4-5 | sed "s/$/ $i/">>$MY/test.txt

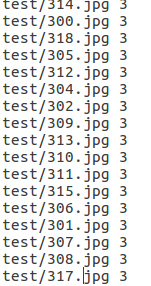
done

echo "All done"

运行脚本：

…/glenn$ sh mycaffe/myimagenet/myconfig/create\_filelist.sh

这样就在myconfig文件夹下生成了两个文本文件，里面是图片的列表清单。



【\*\*\*】

接着再编写一个脚本文件，调用caffe中的conver\_imageset命令来转换数据格式

…/myconfig$sudo vi create\_lmdb.sh

编辑文本：

#!/usr/bin/env sh

MY=mycaffe/myimagenet/myconfig

echo "Create train lmdb.."

rm -rf $MY/img\_train\_lmdb

/home/work/glenn/caffe/build/tools/convert\_imageset \

--shuffle \

--resize\_height=256 \

--resize\_width=256 \

/home/work/glenn/mycaffe/myimagenet/ \

$MY/train.txt \

$MY/img\_train\_lmdb

echo "Create test lmdb.."

rm -rf $MY/img\_test\_lmdb

/home/work/glenn/caffe/build/tools/convert\_imageset \

--shuffle \

--resize\_width=256 \

--resize\_height=256 \

/home/work/glenn/mycaffe/myimagenet/ \

$MY/test.txt \

$MY/img\_test\_lmdb

echo "All Done.."

执行命令：

..glenn$ sudo sh mycaffe/myimagenet/myconfig/create\_lmdb.sh

（3）计算均值并保存

图片减去均值再训练会提高训练速度和精度，因此一般都有该操作，caffe程序中提供了一个计算均值的文件compute\_image\_mean.cpp，直接使用即可：

..glenn$caffe/build/tools/compute\_image\_mean mycaffe/myimagenet/myconfig/img\_train\_lmdb/ mycaffe/myimagenet/myconfig/mean.binaryproto

（4）创建模型并编写配置文件

模型可以用自带的caffenet模型，位置在models/bvlc\_reference\_caffenet文件夹下，将需要的两个配置文件solver.prototxt 和train\_val.prototxt，copy到myconfig文件夹内

修改其中的solver.prototxt

net: "mycaffe/myimagenet/myconfig/train\_val.prototxt"

test\_iter: 2

test\_interval: 100

base\_lr: 0.001

lr\_policy: "step"

gamma: 0.1

stepsize: 1000

display: 20

max\_iter: 5000

momentum: 0.9

weight\_decay: 0.0005

snapshot: 1000

snapshot\_prefix: "mycaffe/myimagenet/myconfig/mycaffenet\_train"

solver\_mode: GPU

100个测试数据，batch\_size为50，因此test\_iter设置为2就能全覆盖。在训练过程中调整学习率逐步变小。

修改train\_val.protxt，修改data layer的mean\_file和source两个地方。

name: "CaffeNet"

layer {

name: "data"

type: "Data"

top: "data"

top: "label"

include {

phase: TRAIN

}

transform\_param {

mirror: true

crop\_size: 227

mean\_file: "mycaffe/myimagenet/myconfig/mean.binaryproto"

}

data\_param {

source: "mycaffe/myimagenet/myconfig/img\_train\_lmdb"

batch\_size: 256

backend: LMDB

}

}

layer {

name: "data"

type: "Data"

top: "data"

top: "label"

include {

phase: TEST

}

transform\_param {

mirror: false

crop\_size: 227

mean\_file: "mycaffe/myimagenet/myconfig/mean.binaryproto"

}

data\_param {

source: "mycaffe/myimagenet/myconfig/img\_test\_lmdb"

batch\_size: 50

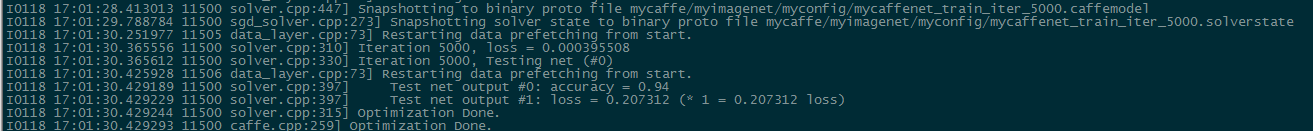
backend: LMDB

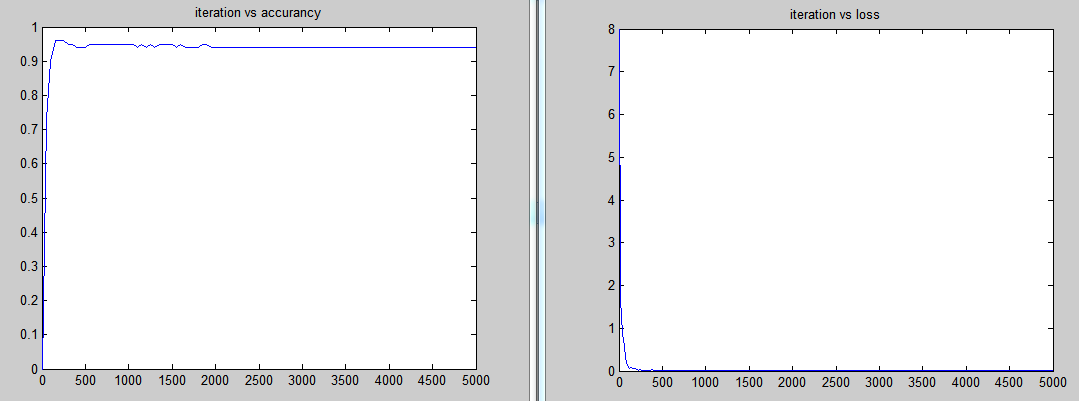
}

}

（5） 训练和测试

..glenn $caffe/build/tools/caffe train -solver mycaffe/myimagenet/myconfig/solver.prototxt





根据测试精度和训练损失，可以看出在100-250次迭代后，就基本到达了训练的最好效果。

精度基本在95%，损失值在4\*10-3左右。

（6）模型参数调整测试

【学习率调整】

由0.001 改为 0.01.其他参数基本不变。（根据上面的情况，适当减少了测试的最大迭代次数）

net: "mycaffe/myimagenet/myconfig/train\_val.prototxt"

test\_iter: 2

test\_interval: 100

base\_lr: 0.01

lr\_policy: "step"

gamma: 0.1

stepsize: 200

display: 20

max\_iter: 500

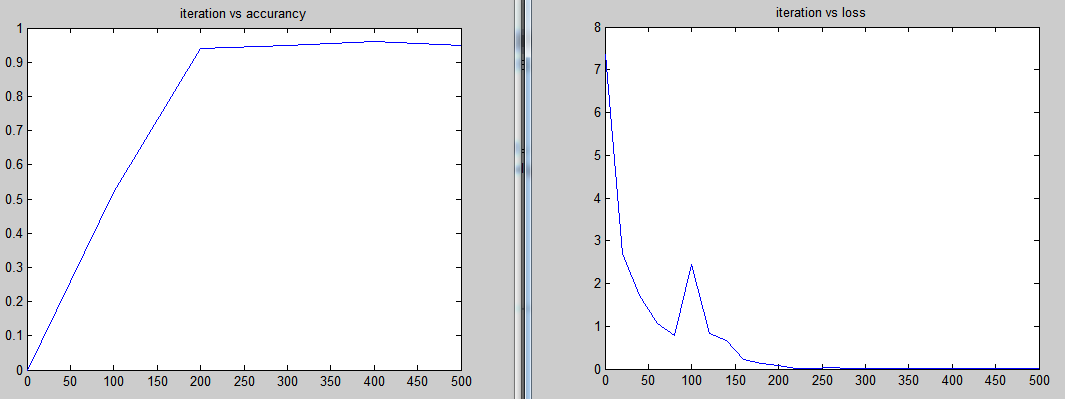
momentum: 0.9

weight\_decay: 0.0005

snapshot: 250

snapshot\_prefix: "mycaffe/myimagenet/myconfig/mycaffenet\_train"

solver\_mode: GPU



改为 0.0001.其他参数基本不变。

net: "mycaffe/myimagenet/myconfig/train\_val.prototxt"

test\_iter: 2

test\_interval: 100

base\_lr: 0.0001

lr\_policy: "step"

gamma: 0.1

stepsize: 200

display: 20

max\_iter: 1500

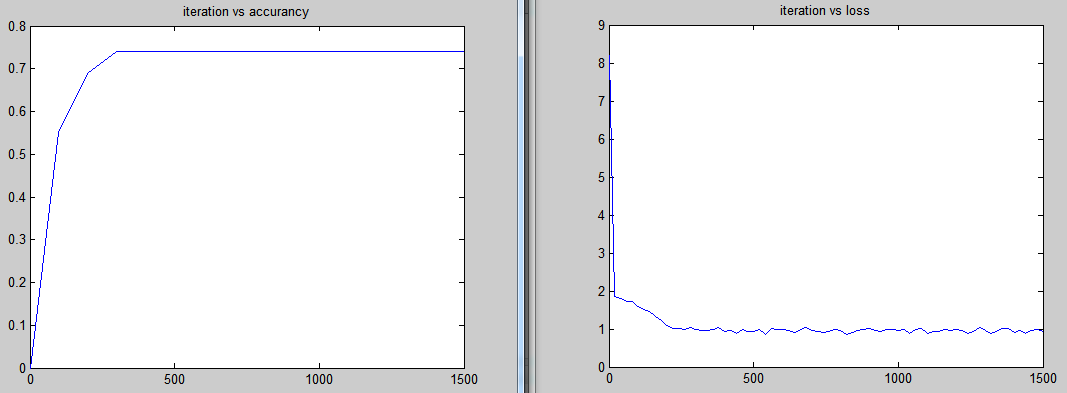
momentum: 0.9

weight\_decay: 0.0005

snapshot: 250

snapshot\_prefix: "mycaffe/myimagenet/myconfig/mycaffenet\_train"

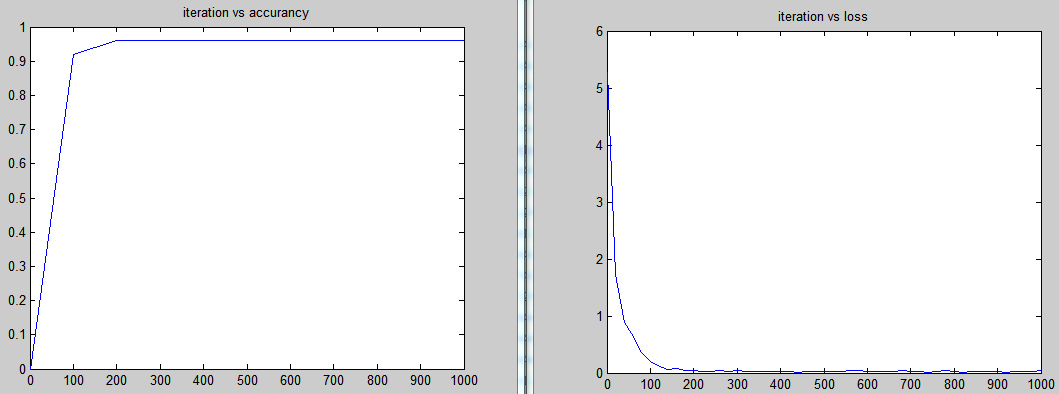
solver\_mode: GPU



从上面三种学习率大小来看，过大或者过小的学习率都不利于得到好的训练效果。

【更改网络结构】

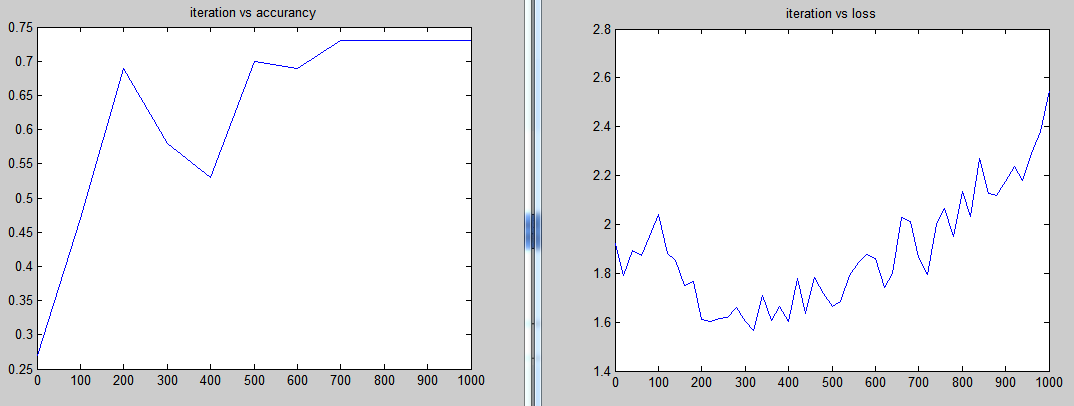
由于原始的Alexnet输出是1000维，对应了1000个类。由于这里只有5类，所以首先修改输出的维度，首先改为100维。



可以看出，训练的效果和精度都有提高。

再将输出改为5维，发下结果很奇怪，精度很快到1，但是损失函数却是87，值很大。

再将学习率调低为0.0001，输出层依旧为5维，结果如下：



测试了不同的学习率，依旧效果不是很理想，有可能是全连接层的参数不是很合适。

根据测试，更改最后输出层和全连接层的网络结构以后，训练效果比较难到达很好的效果。而且同一种网络，多次重复训练的效果也不一定相同，训练达到的精度也不一定一样。可能是每次初始化权重时候，如果初始化值比较好，训练效果会稍好，否则训练效果容易陷入局部最优解。

6.2 基于Alexnet网络结果，以及imagenet1000数据集进行训练和测试

这里利用官方数据集训练，命令如下：

./build/tools/caffe train --solver=models/bvlc\_reference\_caffenet/solver.prototxt

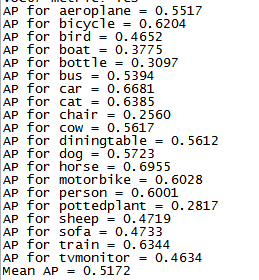
## 基于VOC2007的faster rcnn目标检测模型训练

该题目使用了两种方法进行训练和测试，分别是：

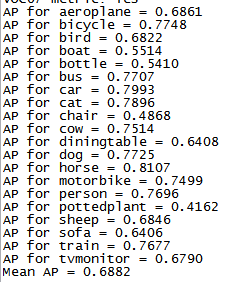
（1）ZF网络+alt\_opt训练算法

（2）VGG16网络+end2end训练算法

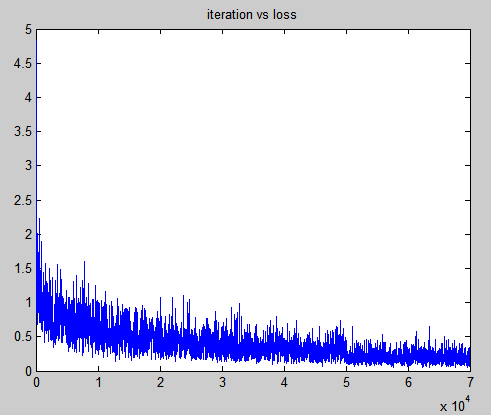
基于第一种方法，最后模型的测试结果如下：



基于第二种方法，最后模型的测试结果如下：



可以看出VGG16网络方案的训练效果要优于ZF网络方案。对VGG16训练过程的损失进行可视化处理，结果如下图：



利用VGG16网络进行sample图片的测试，利用tools/demo.py文件，结果如下：

