# [Deeplearning Toolbox中CNN代码学习](http://blog.csdn.net/sheng_ai/article/details/24133839)

    最近在研究CNN的实现，在还没自己动手写之前，先看看现在比较常用的Matlab的Deeplearning Toolbox中的CNN实现。

    在Toolbox中，CNN的实现过程主要包括下面几个文件

    cnntrain.m           //用于训练CNN的m文件

    cnntest.m            //用于预测的m文件

    cnnsetup.m           //设置训练参数的m文件

    cnnnumgradcheck.m    //用于梯度检查的m文件

    cnnff.m              //用于计算前向过程的m文件

    cnnbp.m              //用于计算误差方向传递的m文件

    cnnapplygrads.m      //用于使用梯度下降算法的m文件

    在test\_example\_CNN.m中，Toolbox的作者使用了MNIST手写字体数据库对CNN部分进行了测试。我将从test\_example\_CNN.m文件开始，按照使用CNN的过程对CNN的每一部分代码进行学习。

**[plain]** [view plaincopyprint?](http://blog.csdn.net/sheng_ai/article/details/24133839)[在CODE上查看代码片](https://code.csdn.net/snippets/302103)

1. test\_example\_CNN.m
3. function test\_example\_CNN
4. load mnist\_uint8;
6. % 将样本调整大小，并进行归一化
7. train\_x = double(reshape(train\_x',28,28,60000))/255;
8. test\_x = double(reshape(test\_x',28,28,10000))/255;
9. train\_y = double(train\_y');
10. test\_y = double(test\_y');
12. %% ex1 Train a 6c-2s-12c-2s Convolutional neural network
13. %will run 1 epoch in about 200 second and get around 11% error.
14. %With 100 epochs you'll get around 1.2% error
16. rand('state',0)

19. % CNN的网络结构，这些只是说明卷积层和下采样层，CNN是一个结构数组
20. cnn.layers = {
21. struct('type', 'i') %input layer
22. struct('type', 'c', 'outputmaps', 6, 'kernelsize', 5) %convolution layer
23. struct('type', 's', 'scale', 2) %sub sampling layer
24. struct('type', 'c', 'outputmaps', 12, 'kernelsize', 5) %convolution layer
25. struct('type', 's', 'scale', 2) %subsampling layer
26. };

29. opts.alpha = 1;        %BP算法的学习速率
30. opts.batchsize = 50;   %进行一个训练的样本的大小
31. opts.numepochs = 5;    %迭代次数

34. cnn = cnnsetup(cnn, train\_x, train\_y);        %设置CNN的结构
35. cnn = cnntrain(cnn, train\_x, train\_y, opts);  %训练CNN
37. [er, bad] = cnntest(cnn, test\_x, test\_y);     %测试CNN
39. %plot mean squared error
40. figure; plot(cnn.rL);
41. assert(er<0.12, 'Too big error');
43. er                   %显示错误率
44. save('CNN-10');      %保存神经网络

    从test\_example\_CNN.m文件可以看出，这个文件主要按照下面的步骤进行：

1. 将读取训练样本和测试样本；
2. 将样本恢复为图像，并归一化为0~1；
3. 设置CNN网络的层数，每一层的结构，学习速率，batch的大小，迭代的次数；
4. 调用cnnsetup函数，初始化CNN的结构和参数；
5. 调用cnntrain函数，训练CNN；
6. 调用cnntest函数，对测试样本进行预测；
7. 输出错误率。

**[plain]** [view plaincopyprint?](http://blog.csdn.net/sheng_ai/article/details/24133839)[在CODE上查看代码片](https://code.csdn.net/snippets/302103)

1. cnnsetup.m
2. function net = cnnsetup(net, x, y)
3. %    assert(~isOctave() || compare\_versions(OCTAVE\_VERSION, '3.8.0', '>='), ['Octave 3.8.0 or greater is required for CNNs as there is a bug in convolution in previous versions. See http://savannah.gnu.org/bugs/?39314. Your version is ' OCTAVE\_VERSION]);
5. % 每一层输入的特征图的个数
6. inputmaps = 1;
7. mapsize = size(squeeze(x(:, :, 1)));
9. for l = 1 : numel(net.layers)   %  layer
11. if strcmp(net.layers{l}.type, 's')   %下采样层
12. mapsize = mapsize / net.layers{l}.scale;  %修改特征图的大小
13. assert(all(floor(mapsize)==mapsize), ['Layer ' num2str(l) ' size must be integer. Actual: ' num2str(mapsize)]);
14. for j = 1 : inputmaps
15. net.layers{l}.b{j} = 0;  %下采样层的偏移初始化为0，此处并没有安装经典的LeNet-5中设置一个系数参数
16. end
17. end
19. if strcmp(net.layers{l}.type, 'c')  %卷积层
20. mapsize = mapsize - net.layers{l}.kernelsize + 1;  %卷积层特征图大小
21. fan\_out = net.layers{l}.outputmaps \* net.layers{l}.kernelsize ^ 2;
22. for j = 1 : net.layers{l}.outputmaps  %  output map
23. fan\_in = inputmaps \* net.layers{l}.kernelsize ^ 2;
24. for i = 1 : inputmaps  %  input map
25. net.layers{l}.k{i}{j} = (rand(net.layers{l}.kernelsize) - 0.5) \* 2 \* sqrt(6 / (fan\_in + fan\_out));  %卷积核参数随机初始化，此处为矩阵
26. end
27. net.layers{l}.b{j} = 0;   %卷积层的偏置初始化为0
28. end
29. inputmaps = net.layers{l}.outputmaps; %将下一层的输入特征图个数更新为此卷积层的输出特征图个数
30. end
31. end
33. % 'onum' is the number of labels, that's why it is calculated using size(y, 1). If you have 20 labels so the output of the network will be 20 neurons.
34. % 'fvnum' is the number of output neurons at the last layer, the layer just before the output layer.
35. % 'ffb' is the biases of the output neurons.
36. % 'ffW' is the weights between the last layer and the output neurons. Note that the last layer is fully connected to the output layer, that's why the size of the weights is (onum \* fvnum)
37. fvnum = prod(mapsize) \* inputmaps;
38. onum = size(y, 1);     %输出层的神经元个数
40. net.ffb = zeros(onum, 1);  %初始化为0
41. net.ffW = (rand(onum, fvnum) - 0.5) \* 2 \* sqrt(6 / (onum + fvnum)); %随机初始化
42. end

    从cnnsetup.m文件可以看出，CNN结构中每一层的特征图的输入特征图的个数，以及每一层的可训练参数都在cnnsetup函数中进行设置。这里和经典的LeNet-5中使用的CNN结构不同之处在于，对于下采样层，并没有设置可训练的系数参数。

cnntrain.m文件

**[plain]** [view plaincopyprint?](http://blog.csdn.net/sheng_ai/article/details/24133839)[在CODE上查看代码片](https://code.csdn.net/snippets/302103)

1. function net = cnntrain(net, x, y, opts)
2. % This function is used to train the CNN
3. m = size(x, 3);   %训练样本个数，每次迭代都使用所有样本
4. numbatches = m / opts.batchsize;  %样本包个数，在一个包的样本进行梯度计算后才更新参数
6. if rem(numbatches, 1) ~= 0        %样本包个数不是常数时报错
7. error('numbatches not integer');
8. end
10. net.rL = [];
12. for i = 1 : opts.numepochs   %开始进行迭代
13. disp(['epoch ' num2str(i) '/' num2str(opts.numepochs)]);
14. tic;  %计时开始
16. kk = randperm(m);
17. for l = 1 : numbatches
19. %随机抽取batchsize个训练样本
20. batch\_x = x(:, :, kk((l - 1) \* opts.batchsize + 1 : l \* opts.batchsize));
21. batch\_y = y(:,    kk((l - 1) \* opts.batchsize + 1 : l \* opts.batchsize));
23. net = cnnff(net, batch\_x);   %计算CNN的forward的值
24. net = cnnbp(net, batch\_y);   %计算CNN的梯度？
25. net = cnnapplygrads(net, opts);  %使用梯度下降算法训练CNN
26. if isempty(net.rL)
27. net.rL(1) = net.L;
28. end
29. net.rL(end + 1) = 0.99 \* net.rL(end) + 0.01 \* net.L;
30. end
31. toc;
32. end
34. end

   在cnntrain.m文件中，主要是调用了cnnff进行前向计算，然后调用cnnbp函数进行误差计算，最后调用cnnappplygrads函数更新cnn的参数。

cnnff.m文件

**[plain]** [view plaincopyprint?](http://blog.csdn.net/sheng_ai/article/details/24133839)[在CODE上查看代码片](https://code.csdn.net/snippets/302103)

1. function net = cnnff(net, x)
2. %This function is used to calculateing the forward
3. n = numel(net.layers);
4. net.layers{1}.a{1} = x;
5. inputmaps = 1;
7. for l = 2 : n   %  for each layer
9. if strcmp(net.layers{l}.type, 'c')     %卷积层
10. %  !!below can probably be handled by insane matrix operations
11. for j = 1 : net.layers{l}.outputmaps   %  for each output map
12. %  create temp output map
13. z = zeros(size(net.layers{l - 1}.a{1}) - [net.layers{l}.kernelsize - 1 net.layers{l}.kernelsize - 1 0]);
15. for i = 1 : inputmaps   %  for each input map    %所有输入层都进行卷积并相加？
16. %  convolve with corresponding kernel and add to temp output map
17. z = z + convn(net.layers{l - 1}.a{i}, net.layers{l}.k{i}{j}, 'valid');
18. end
19. %  add bias, pass through nonlinearity
20. net.layers{l}.a{j} = sigm(z + net.layers{l}.b{j});  %上采样层经过Sigmoid函数进行激活
21. end
23. %  set number of input maps to this layers number of outputmaps
24. inputmaps = net.layers{l}.outputmaps;

27. elseif strcmp(net.layers{l}.type, 's')  % 下采样层没有和乘以一个系数和加上一个偏置
28. %  downsample
29. for j = 1 : inputmaps
30. z = convn(net.layers{l - 1}.a{j}, ones(net.layers{l}.scale) / (net.layers{l}.scale ^ 2), 'valid');   %  !! replace with variable
31. net.layers{l}.a{j} = z(1 : net.layers{l}.scale : end, 1 : net.layers{l}.scale : end, :);             % 下采样层没有经过Sigmoid函数激活？
32. end
33. end
34. end

37. %  concatenate all end layer feature maps into vector
38. % 将最后一层特征图展开为列向量的形式
39. net.fv = [];
40. for j = 1 : numel(net.layers{n}.a)
41. sa = size(net.layers{n}.a{j});
42. net.fv = [net.fv; reshape(net.layers{n}.a{j}, sa(1) \* sa(2), sa(3))];
43. end
45. %  feedforward into output perceptrons
46. net.o = sigm(net.ffW \* net.fv + repmat(net.ffb, 1, size(net.fv, 2)));  % 输出层为普通sigm激活？
48. end

    在cnnff.m文件中，我们可以看出，对于下采样层，这个实现并没有使用到在cnnsetup中初始化的偏置b，同时，下采样层也没有经过sigmoid函数进行激活。这个与经典的LeNet-5的实现之间有较大区别，同时，在这个实现中，对于卷积层的每一个特征图，都是将前一层中所有的特征图分别进行卷积后全部相加，然后加上一个偏置后，经过Sigmoid函数激活后得到的，并没有经过选择前一层的某些特征图进行卷积的过程。这一点和LeNet-5的实现也有所不同。

cnnbp.m

**[plain]** [view plaincopyprint?](http://blog.csdn.net/sheng_ai/article/details/24133839)[在CODE上查看代码片](https://code.csdn.net/snippets/302103)

1. function net = cnnbp(net, y)
2. n = numel(net.layers);
4. %   error
5. net.e = net.o - y;
6. %  loss function
7. net.L = 1/2\* sum(net.e(:) .^ 2) / size(net.e, 2);
9. %%  backprop deltas
10. net.od = net.e .\* (net.o .\* (1 - net.o));   %  output delta
11. net.fvd = (net.ffW' \* net.od);              %  feature vector delta
12. if strcmp(net.layers{n}.type, 'c')         %  only conv layers has sigm function
13. net.fvd = net.fvd .\* (net.fv .\* (1 - net.fv));
14. end
16. %  reshape feature vector deltas into output map style
17. sa = size(net.layers{n}.a{1});
18. fvnum = sa(1) \* sa(2);
19. for j = 1 : numel(net.layers{n}.a)
20. net.layers{n}.d{j} = reshape(net.fvd(((j - 1) \* fvnum + 1) : j \* fvnum, :), sa(1), sa(2), sa(3));
21. end
23. % 非输出层的delta计算
24. for l = (n - 1) : -1 : 1
26. % 卷积层的的delta的计算
27. if strcmp(net.layers{l}.type, 'c')
28. for j = 1 : numel(net.layers{l}.a)
29. net.layers{l}.d{j} = net.layers{l}.a{j} .\* (1 - net.layers{l}.a{j}) .\* (expand(net.layers{l + 1}.d{j}, [net.layers{l + 1}.scale net.layers{l + 1}.scale 1]) / net.layers{l + 1}.scale ^ 2);
30. end
32. % 下采样层的delta计算
33. elseif strcmp(net.layers{l}.type, 's')
34. for i = 1 : numel(net.layers{l}.a)
35. z = zeros(size(net.layers{l}.a{1}));
36. for j = 1 : numel(net.layers{l + 1}.a)
37. z = z + convn(net.layers{l + 1}.d{j}, rot180(net.layers{l + 1}.k{i}{j}), 'full');
38. end
39. net.layers{l}.d{i} = z;
40. end
41. end
42. end
44. %%  calc gradients
45. for l = 2 : n
46. if strcmp(net.layers{l}.type, 'c')  %卷积层进行梯度计算，为什么下采样层不用进行梯度计算？
47. for j = 1 : numel(net.layers{l}.a)
48. for i = 1 : numel(net.layers{l - 1}.a)
49. net.layers{l}.dk{i}{j} = convn(flipall(net.layers{l - 1}.a{i}), net.layers{l}.d{j}, 'valid') / size(net.layers{l}.d{j}, 3);
50. end
51. net.layers{l}.db{j} = sum(net.layers{l}.d{j}(:)) / size(net.layers{l}.d{j}, 3);
52. end
53. end
54. end
56. net.dffW = net.od \* (net.fv)' / size(net.od, 2);
57. net.dffb = mean(net.od, 2);
59. function X = rot180(X)
60. X = flipdim(flipdim(X, 1), 2);
61. end
62. end

    在cnnbp.m文件中，主要计算了各层之间的delta，以及卷积层的参数的梯度，由于在cnnff函数中，没有用到下采样层的偏置，所以，在cnnbp函数中，并没有计算该参数的梯度的过程。

cnnapplaygrads.m文件

**[plain]** [view plaincopyprint?](http://blog.csdn.net/sheng_ai/article/details/24133839)[在CODE上查看代码片](https://code.csdn.net/snippets/302103)

1. function net = cnnapplygrads(net, opts)
2. % 使用梯度下降算法进行参数训练
3. for l = 2 : numel(net.layers)
4. if strcmp(net.layers{l}.type, 'c')  %只更新卷积层的参数？没有更新下采样层的参数？
5. for j = 1 : numel(net.layers{l}.a)
6. for ii = 1 : numel(net.layers{l - 1}.a)
7. net.layers{l}.k{ii}{j} = net.layers{l}.k{ii}{j} - opts.alpha \* net.layers{l}.dk{ii}{j};
8. end
9. net.layers{l}.b{j} = net.layers{l}.b{j} - opts.alpha \* net.layers{l}.db{j};
10. end
11. end
12. end
14. net.ffW = net.ffW - opts.alpha \* net.dffW;
15. net.ffb = net.ffb - opts.alpha \* net.dffb;
16. end

    在cnnapplaygrads函数中，主要是使用梯度下降算法更新卷积层的参数和输出层的参数，由于在cnnff中并没有使用到下采样层的偏置这个参数，所以在cnnapplaygrads函数中，也没有更新这个参数的过程。

cnntest.m文件

**[plain]** [view plaincopyprint?](http://blog.csdn.net/sheng_ai/article/details/24133839)[在CODE上查看代码片](https://code.csdn.net/snippets/302103)

1. function [er, bad] = cnntest(net, x, y)
2. %  feedforward
3. net = cnnff(net, x);
5. % CNN给出的样本的输出的类别，此类别为输出神经元中最大的部分，此部分与传统神经BP神经网络一致
6. [~, h] = max(net.o);
8. % 样本的实际输出类别
9. [~, a] = max(y);
11. % 寻找预测错误的样本
12. bad = find(h ~= a);
14. er = numel(bad) / size(y, 2);
15. end

    cnntest函数主要用来对测试样本进行预测，并计算错误率，其实，对于CNN来说，对样本进行预测就是简单地使用cnnff函数计算样本的前向过程，然后找出输出层中值最大的那个神经元的类别，就是样本对应的类别，所以，在cnntest函数中，并没有什么重要的内容。

    至此，Toolbox中cnn部分的代码已经介绍完毕，其实，要想能真正理解cnn的训练过程，还是需要看看《Notes on Convolutional Neural Networks》，里面对于CNN的训练过程的介绍还是相当清楚。

    在看完CNN的代码之后，就可以开始动手修改这些代码了。Enjoy it~

# [Deeplearning Toolbox中CNN代码的修改--增加下采样层偏置](http://blog.csdn.net/sheng_ai/article/details/24527059)

    前几天学习了Deeplearning Toolbox中关于CNN的那部分代码，发现这个Toolbox的关于CNN的实现与经典的LeNet-5的实现有下面几点不同：

1. 在下采样层，在计算前一层的神经元之和后没有使用到偏置；
2. 在下采样层，在计算前一层的神经元之和后没有使用到系数；
3. 在下采样层，没有使用Sigmoid函数进行激活

    因此，在学习了CNN的代码后，就要进行修改了。

    在Toolbox中，CNN主要有四个部分组成，cnnsetup.m用于设置CNN的结构和初始化参数，cnnff用于计算前向过程，cnnbp用于计算各层的delta和每一个参数的梯度，cnnapplaygrads使用梯度下降算法计算更新每一个参数。

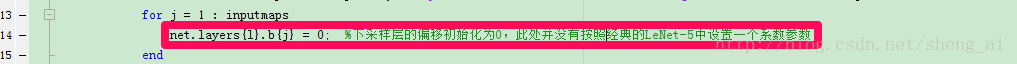
    因此，每添加一个参数或者添加激活函数，都需要在相应的修改者四个文件中的部分。主要步骤如下：

1. 在cnnsetup.m文件中添加相应的参数初始值；
2. 在cnnff.m文件中添加该参数在前向过程中需要加入的计算；
3. 在cnnbp.m文件中添加该参数对于delta计算的部分，已经该参数的梯度计算部分；
4. 在cnnapplaygrads.m文件中添加更新该参数的部分。

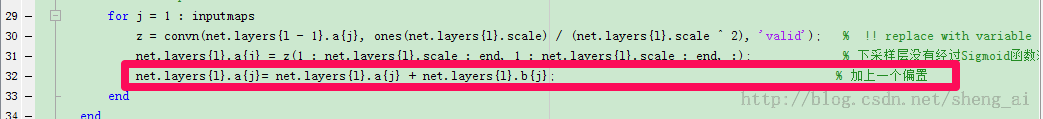
    根据《Notes on Convolutional Neural Networks》中，关于CNN的下采样层的偏置和推导，可以知道，单独添加下采样层并不会影响到下一层和前一层的delta的计算，也不会影响其他参数的梯度的计算，比较容易实现，因此，考虑添加下采样层偏置。

一.增加下采样层的偏置

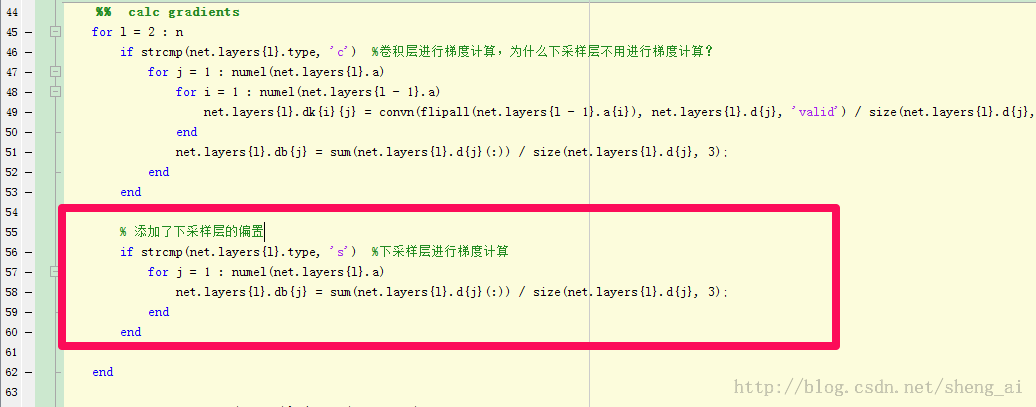
    在cnnsetup.m文件中，已经初始化了下采样层的偏置：



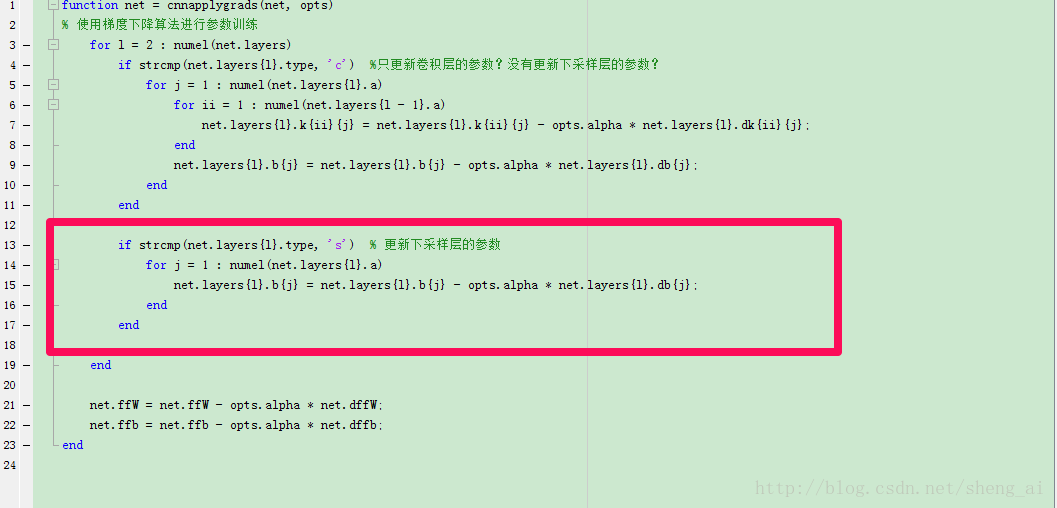
    在cnnff.m文件中，添加下面的代码：



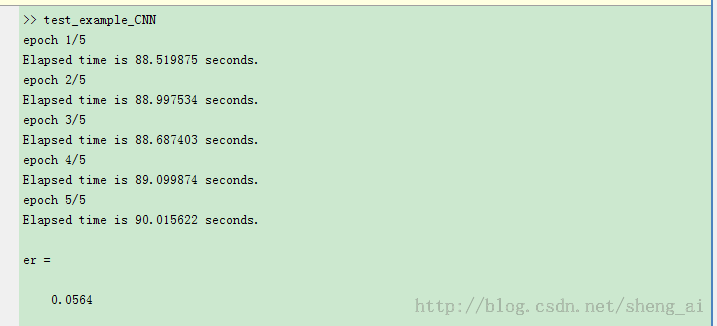
    在cnnbp.m文件中，添加下面的代码



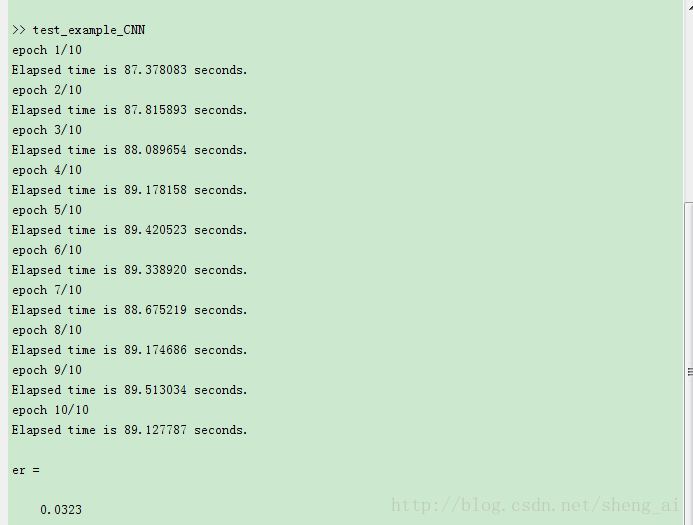
    在cnnapplaygrads.m文件中，添加下面的代码：



     在进行5次的迭代之后，出现的结果如下：



     进行10迭代后的结果如下：



   下一步就是增加下采样层的系数。