ASSIGNMENT A7

Shantnu Kakkar

CS 6320, Spring 2016

March 22, 2016

Section 1: Intro:

This assignment is based on the texture analysis algorithms learnt in the lecture. I will be implementing **algorithm 6.1 and 6.2** from the text. We will be studying two kinds of texture representations: **Local Texture Representations** and **Pooled texture representation**. The former encodes the texture very close to a point in the image, and the latter require a description of the texture within an image domain. Local texture representations are done via filters such as spot filters, bar filters, oriented filters, mean filters, etc. We try to identify repeated elements of an image called Textons, which are made from generic sub elements called spots and bars. The following figure shows a set of 48 oriented filters used for expanding images into a series of responses for texture representations. Figure 2 gives an example of what a local texture representation tries to achieve.

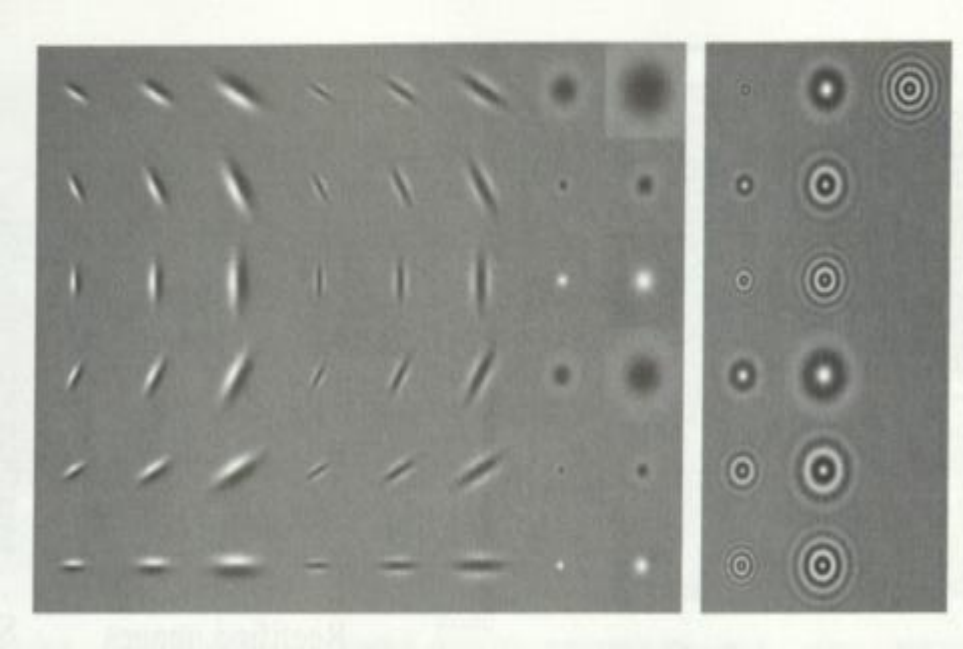


Figure 1 Various oriented filters. Right side shows orientation independent filters

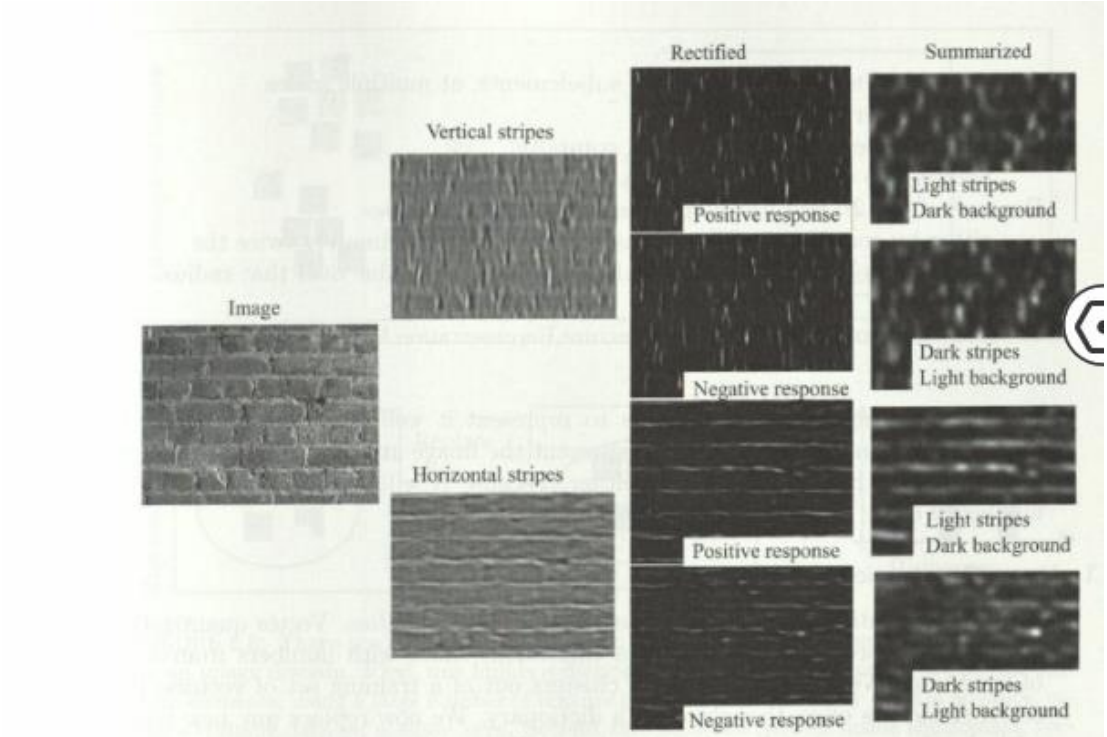


FIGURE 6.7: Filter-based texture representations look for pattern subelements such as oriented bars. The brick image on the **left** is filtered with an oriented bar filter (shown as a tiny inset on the top left of the image at full scale) to detect bars, yielding stripe responses (**center left**; negative is dark, positive is light, mid-gray is zero). These are rectified (here we use half-wave rectification) to yield response maps (**center right**; dark is zero, light is positive). In turn, these are summarized (here we smoothed over a neighborhood twice the filter width) to yield the texture representation on the **right**. In this, pixels that have strong vertical bars nearby are light, and others are dark; there is not much difference between the dark and light vertical structure for this image, but there is a real difference between dark and light horizontal structure.

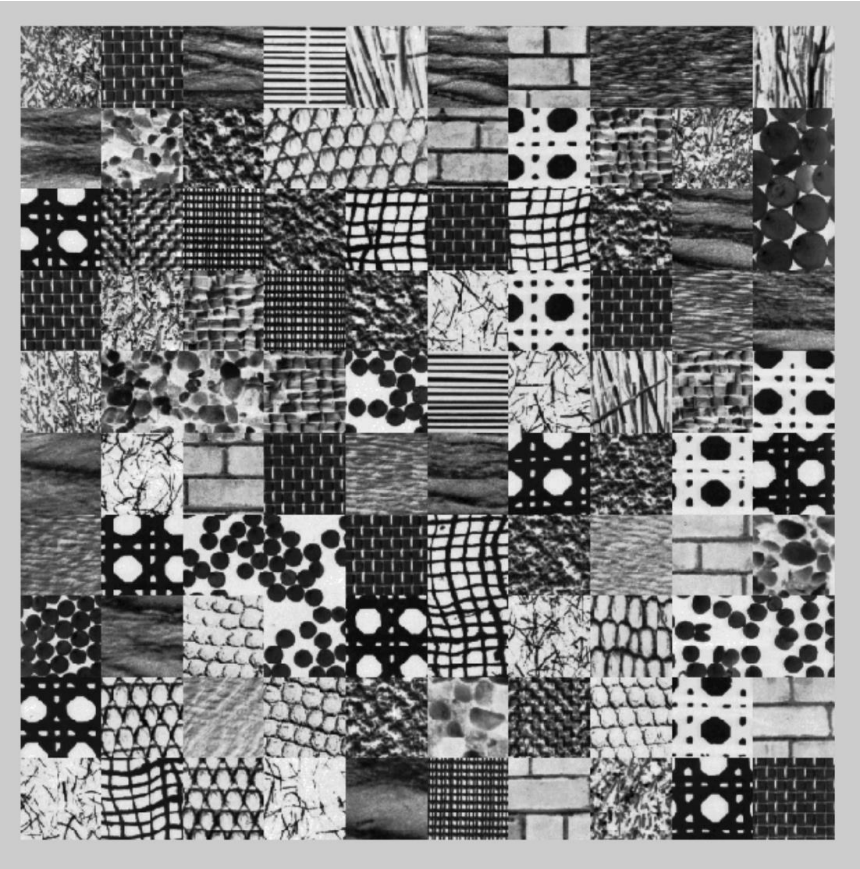
Figure 2 Local texture representation example

Another powerful technique that is commonly used in textures is **vector quantization.** For a repeating patch, there are two important difficulties in finding image patches or vectors if filter outputs that commonly occur together:

1) These representation of the image are continuous. We cannot simply count how many times a particular pattern occurs, because each vector is slightly different.
2) The representation is high dimensional. A patch around a pixel might need hundredths of pixels to represent it well, likewise, hundreds of different might be needed to represent the image at a pixel.

To manage such high number of cells, we use vector quantization. In this technique, we represent vectors in a continuous space with numbers from a set of fixed size. We basically try to represent a collection of vectors as a histogram of cluster centres.

In this assignment, we have been given a mixed texture image (using a .mat file), and we require to develop the Matlab functions implementing (1) spot1 filter, (2) spot2 filter (3) bar filter, (4) image texture parameters, and (5) k means texture. These functions will be used to classify the textures in the given mixed texture. The following image has been given:



We will be doing analyses for various values of k ranging from 4 to 20 by computing statistics on the number of good texture classification vs k

Section 2: Method:

- Matlab is used to carry out the experiments.
- I will be using the combo function provided by professor to analyse my results for CS5320_k_means_texture() functions.
- I will be using CS5320_oriented_Gaussian() function provided by professor to make bar filter.

Following functions are implemented:

- CS5320_spot1(): It helps to make spot 1 texture feature, which is a symmetric gaussian filter. It doesnot take any input and its output is the desired spot filter.
- CS5320_spot2(): It helps to make spot 2 texture feature, which is a symmetric gaussian filter. It doesnot take any input and its output is the desired spot filter.

- CS5320_bar(): It helps to make spot 2 texture feature, which is made from oriented Gaussian filters. It's input is a,b,c,d which are required orientation of the bar, and its output is the bar filter, also called as the bar texture feature.
- CS5320_texture_params: This function takes as input an image, and returns the correspomding texture parameters. This output is 16 channel, and contains positive an dnegative summaries for spot, bar, mean and variance filter. By summary we mean that we convolve the rectified responses with a Gaussian of twice the size of the filters used to produce the parameter images i.e. since 11X11 are used, we do Gaussian at 22X22. However, we will be playing around with the size and sigma of this Gaussian.
- CS5320_k_means_texture: This function classifies textures into clusters. Its inout are the texture parameter images and the number of desired clusters. The outputs are the clusters vectors (which contains k cluster center vectors) and elements(which contains the cluster number). The size of elements vectors is same as that of image. Matlab functions reshape and kmeans are used inside this function. Reshape helps change the size since we want to convert our params from [640,640,16] to [640X640,16].

In addition to the above functions there is a script named "Verification" which contains how I call other function.

The following algorithm has been used for spot1:

- Make symmetric Gaussian G1 = fspecial('gaussian',11,0.62);
- Make symmetric Gaussian G2 = fspecial('gaussian',11,1);
- Make symmetric Gaussian G3 = fspecial('gaussian',11,1.6);
- Using weights as 1,-2, 1, make spot 1 as S1 = G1 - 2*G2 + G3;

The following algorithm has been used for spot2:

- Make symmetric Gaussian G1 = fspecial('gaussian',11,0.71);
- Make symmetric Gaussian G2 = fspecial('gaussian',11,1.14);
- Using weights as 1,-1 make spot 2 as S1 = G1 - G2;

The following algorithm has been used for bar function:

- Define range of x and y values for oriented Gaussian
  xmin = -3;
  xmax = 3;
  ymin = -3;
  ymax = 3;
- Call CS5320_oriented_Gaussian to make three different oriented filters.
  G1 = CS5320_oriented_Gaussian(a,b,c,d,2,1,0,1,xmin,xmax,ymin,ymax,0.1);
  G2 = CS5320_oriented_Gaussian(a,b,c,d,2,1,0,0,xmin,xmax,ymin,ymax,0.1);
  G3 = CS5320_oriented_Gaussian(a,b,c,d,2,1,0,-1,xmin,xmax,ymin,ymax,0.1);
- Make bar using weights -1,2,-1: B = -G1 + 2*G2 - G3;
- Resize your bar B = imresize(B,[11,11]);

The following method is used for finding texture parameter images:

- Initialize params = [];
- Make Gaussian filter which will be used for summary

```
HGaussian = fspecial('gaussian',22,7);
```

- Find spots and bar filters.

```
S1 = CS5320_spot1();
S2 = CS5320_spot2();

B0 = CS5320_bar(1,0,0,-1);
```

- Rotate the bar filter

```
B45 = imrotate(B0,45,'crop');
B90 = imrotate(B0,90,'crop');
B135 = imrotate(B0,135,'crop');
```

- Generate positive and negative summary for spot 1

```
R1 = filter2(S1,im);
map1 = max(0,R1);
map2 = max(0,-R1);
params(:,:,1) = filter2(HGaussian,map1);
params(:,:,2) = filter2(HGaussian,map2);
```

- Generate positive and negative summary for spot 2

```
R1 = filter2(S2,im);
map1 = max(0,R1);
map2 = max(0,-R1);
params(:,:,3) = filter2(HGaussian,map1);
params(:,:,4) = filter2(HGaussian,map2);
```

- Generate positive and negative summary for bar at 0 degrees

```
R1 = filter2(B0,im);
map1 = max(0,R1);
map2 = max(0,-R1);
params(:,:,5) = filter2(HGaussian,map1);
params(:,:,6) = filter2(HGaussian,map2);
```

- Generate positive and negative summary for bar at 45 degrees

```
R1 = filter2(B45,im);
map1 = max(0,R1);
map2 = max(0,-R1);
params(:,:,7) = filter2(HGaussian,map1);
params(:,:,8) = filter2(HGaussian,map2);
```

- Generate positive and negative summary for bar at 90 degrees

```
R1 = filter2(B90,im);
```

```
map1 = max(0,R1);
map2 = max(0,-R1);
params(:,:,9) = filter2(HGaussian,map1);
params(:,:,10) = filter2(HGaussian,map2);
```

- Generate positive and negative summary for bar at 135 degrees

```
R1 = filter2(B135,im);
map1 = max(0,R1);
map2 = max(0,-R1);
params(:,:,11) = filter2(HGaussian,map1);
params(:,:,12) = filter2(HGaussian,map2);
```

- Generate positive and negative summary for mean filter. Here, mean is the average of some window at a pixel

```
HMean = fspecial('average',11);
R1 = filter2(HMean,im);
map1 = max(0,R1);
map2 = max(0,-R1);
params(:,:,13) = filter2(HGaussian,map1);
params(:,:,14) = filter2(HGaussian,map2);
```

- Generate positive and negative summary for variance of window at all the pixel

```
R1 = stdfilt(im,ones(11));
R1 = R1.^2;
map1 = max(0,R1);
map2 = max(0,-R1);
params(:,:,15) = filter2(HGaussian,map1);
params(:,:,16) = filter2(HGaussian,map2);
```

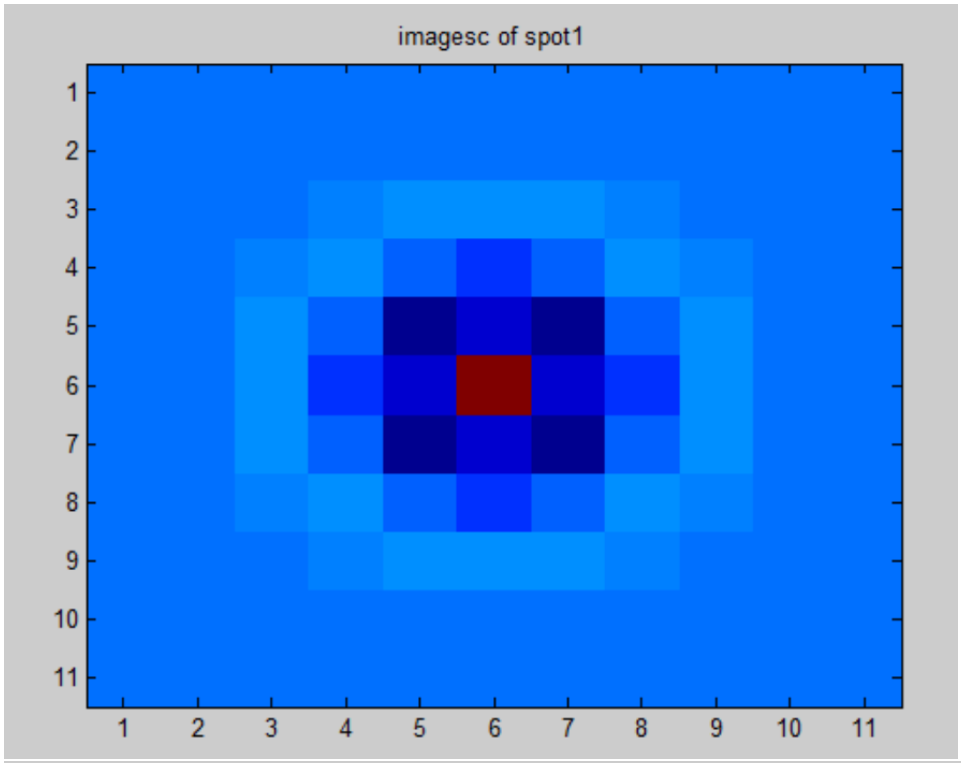
The following method is used for doing the k_means_texture:

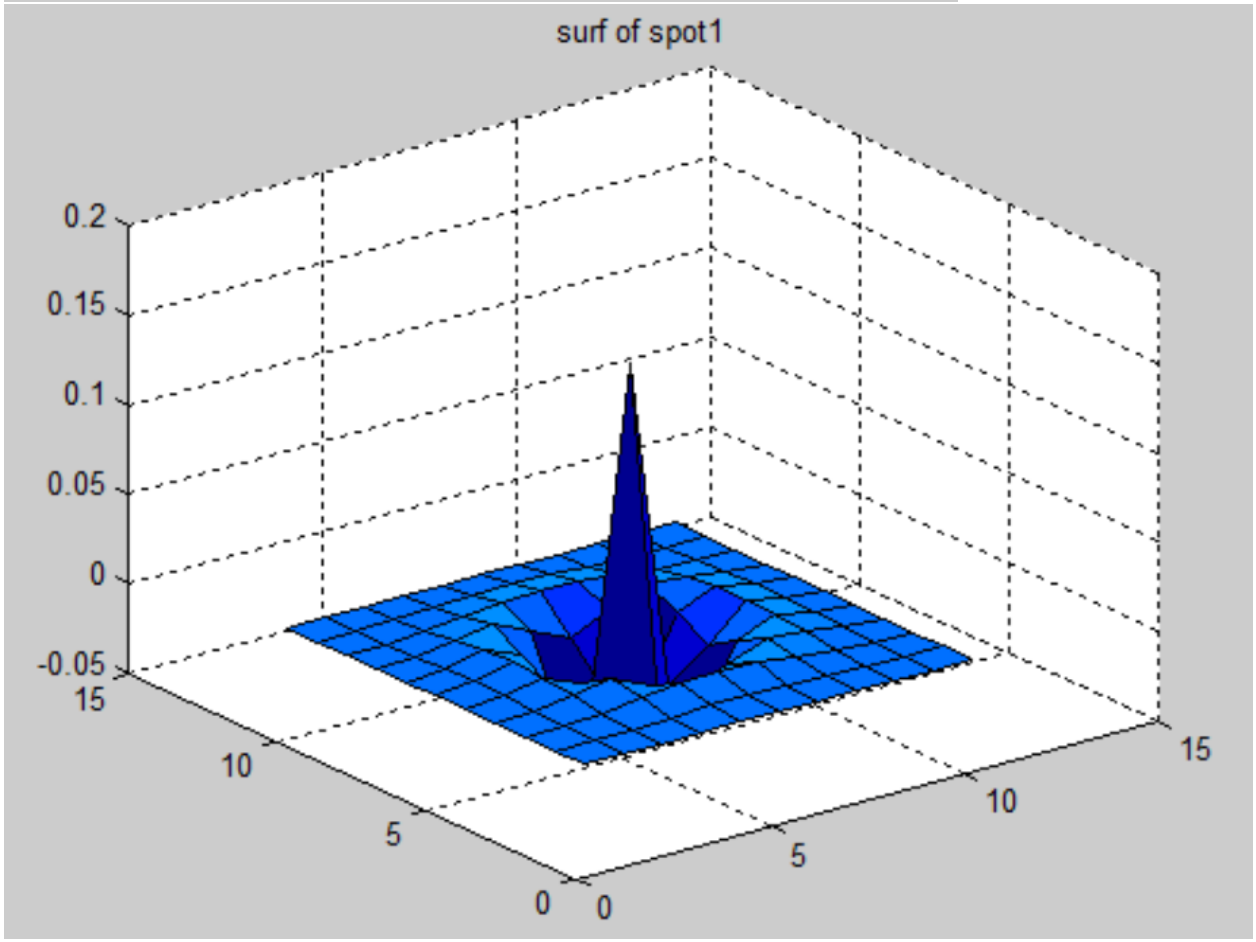
- Find size of texture parameter images [nr,nc,depth] = size(params);
- Reshape parameter images params = reshape(params,[nr*nc,depth]);
- Call kmeans [elements,clusters] = kmeans(params,k);
- Reshape elements = reshape(elements,[nr,nc]);

Section 3: Verification:

Testing CS5320_spot1

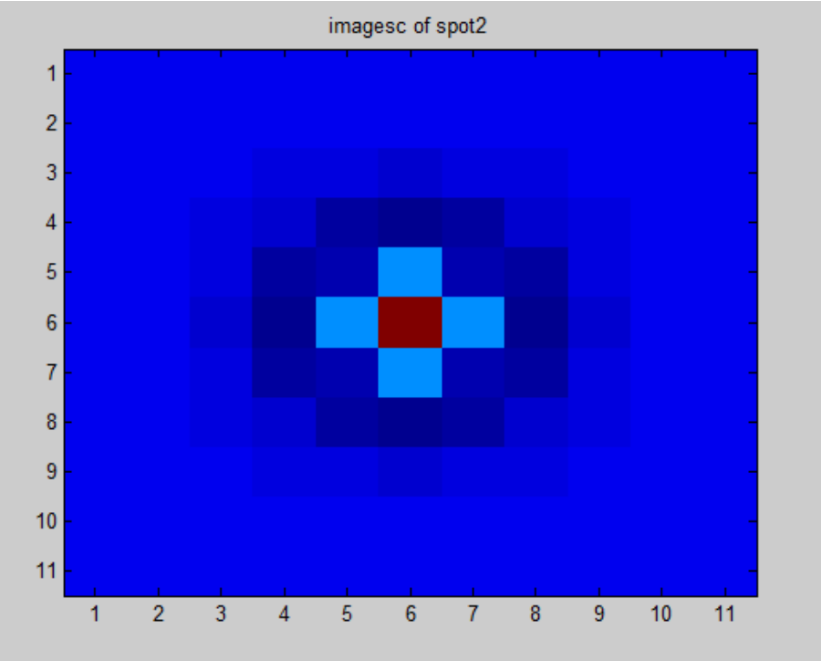
1) **Getting good spot 1 filter as can be seen below.**

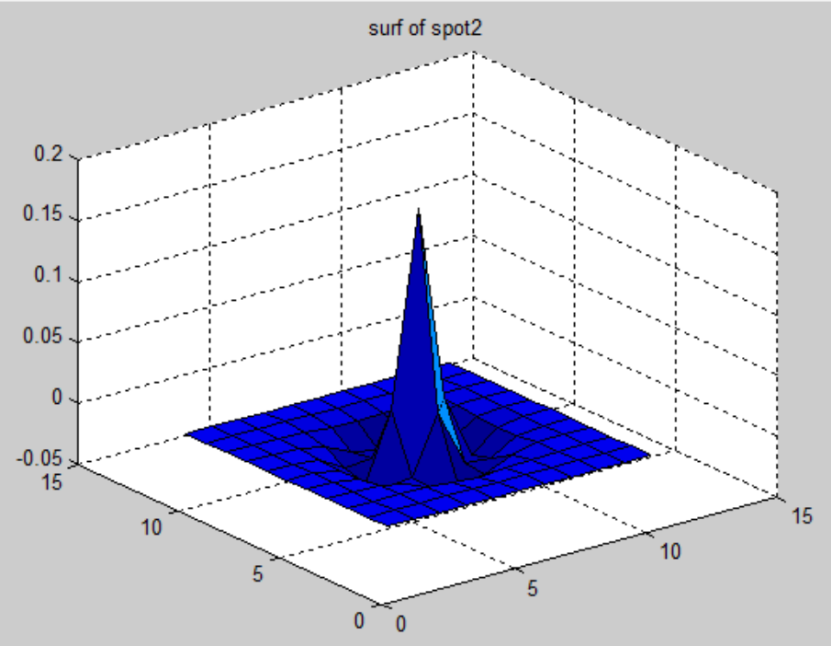

imagesc of spot1
1
2
3
4
5
6
7
8
9
10
11
1
2
3
4
5
6
7
8
9
10
11


surf of spot1
0.2
0.15
0.1
0.05
0
-0.05
15
10
5
0
0
5
10
15

Testing CS5320_spot2

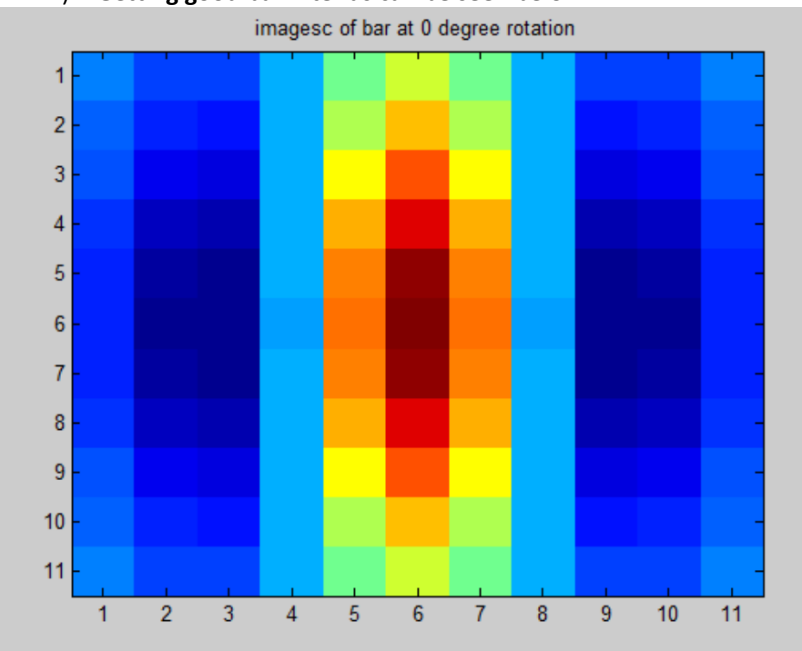
1) **Getting good spot 2 filter as can be seen below.**

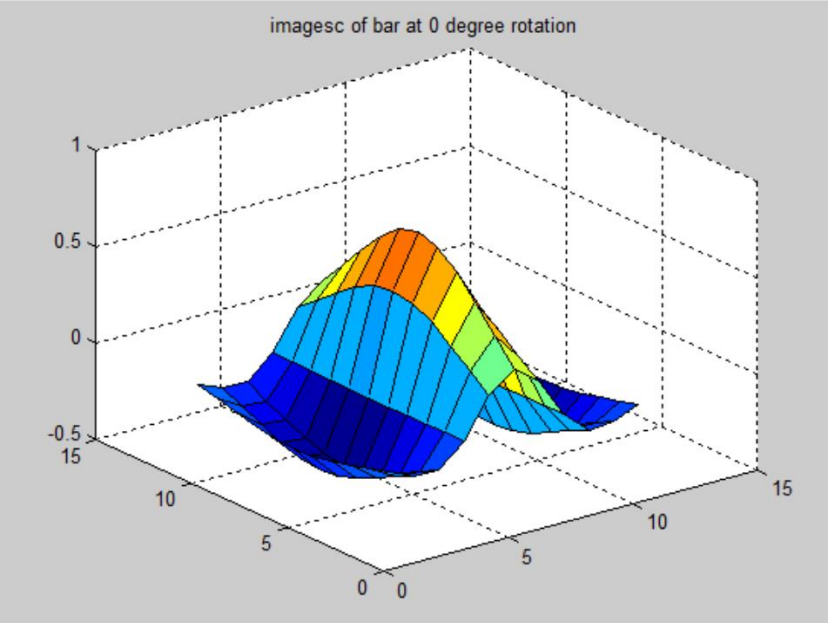




Testing CS5320_bar

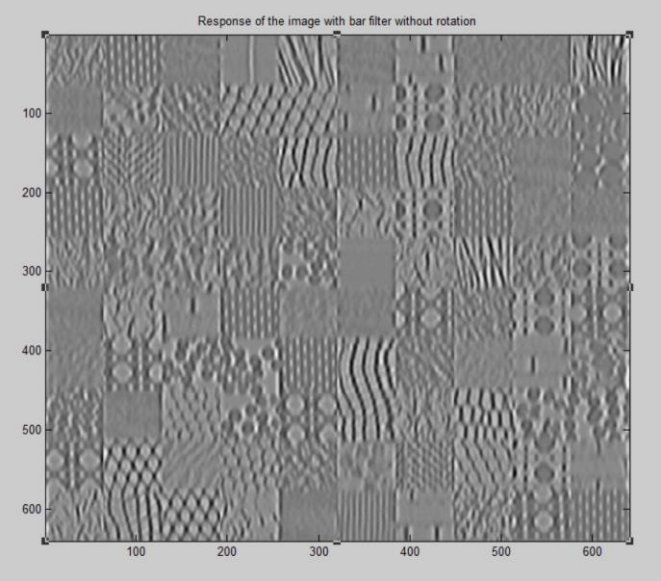
1) **Getting good bar filter as can be seen below.**

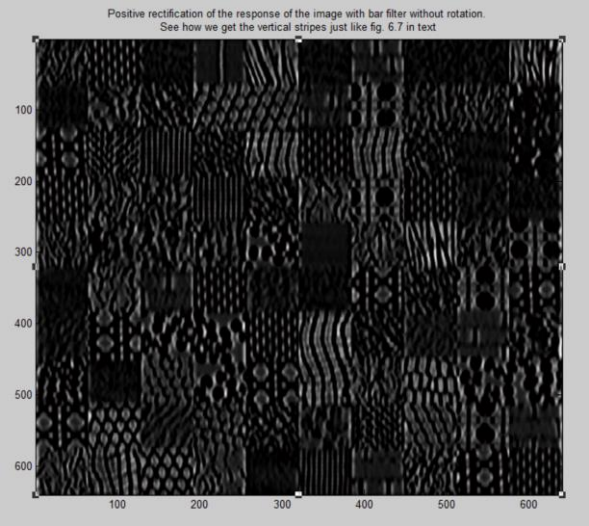


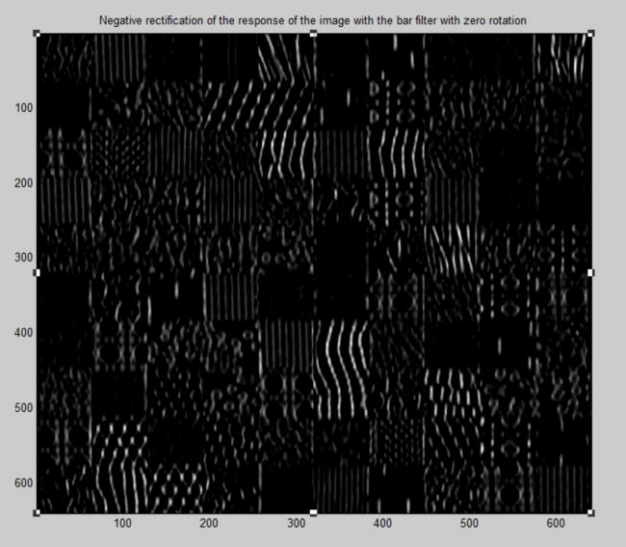


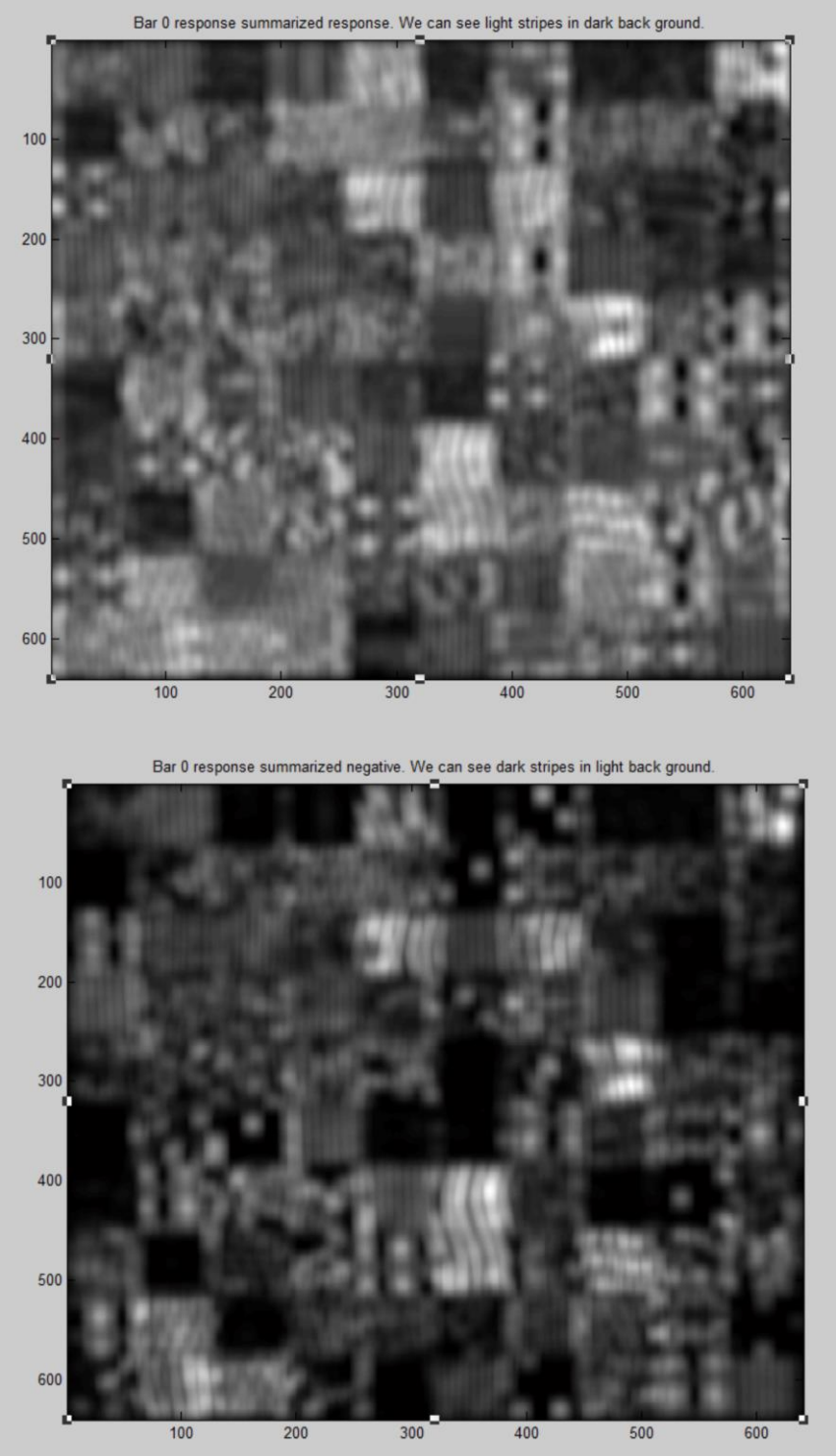
Testing CS5320_texture_params

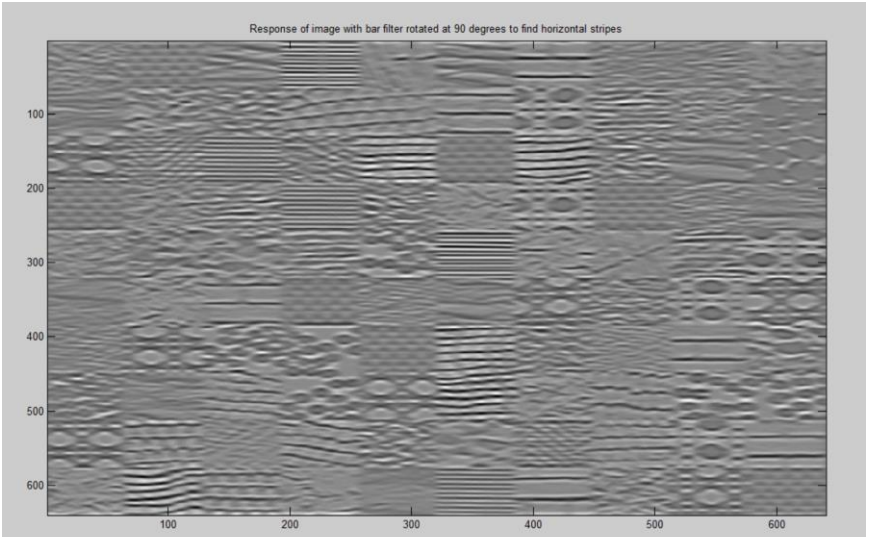
- **To test this function, we do various tests:**
  1) **I am seeing the filtering, rectification and summary produced by my bar filter inside my CS5320_texture_params function. My aim to see whether I will get results similar to figure 6.7 in the text. Using bar filter oriented at zero degrees, I get the vertical stripes and see their rectification and summaries. Likewise, using bar oriented at 90 degrees, I get the horizontal stripes and see their rectification and summaries. The following figures shows the results produced by bar filter inside CS5320_texture_params. I am definitely getting results similar to fig. 6.7 in the text.**

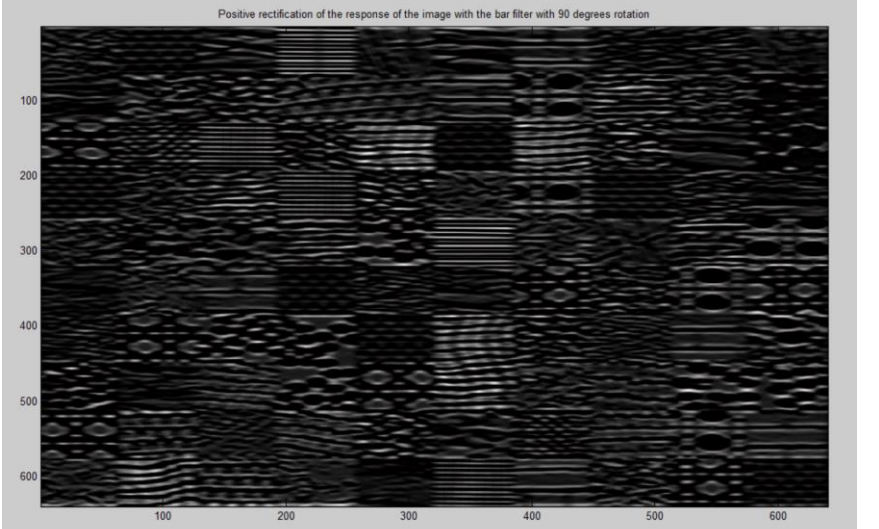

Response of the image with bar filter without rotation
100
200
300
400
500
600
100
200
300
400
500
600

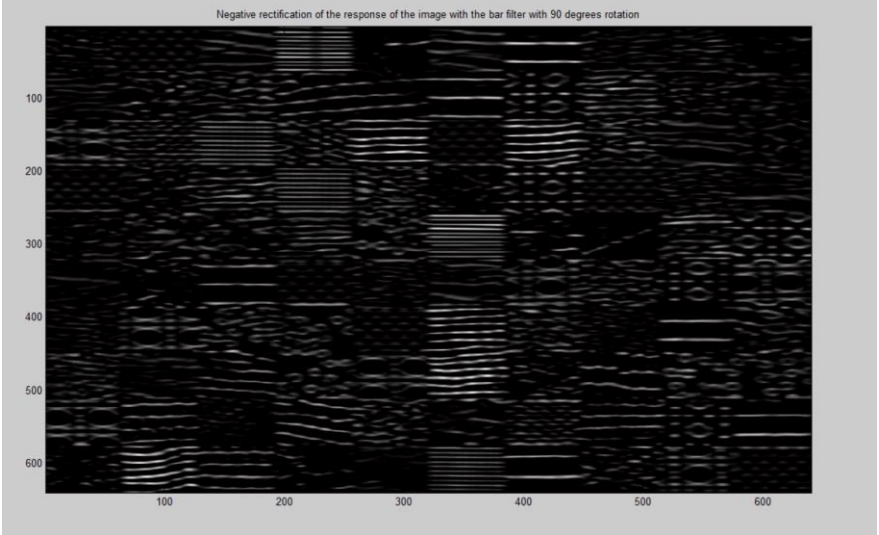

Positive rectification of the response of the image with bar filter without rotation.
See how we get the vertical stripes just like fig. 6.7 in text
100
200
300
400
500
600
100
200
300
400
500
600

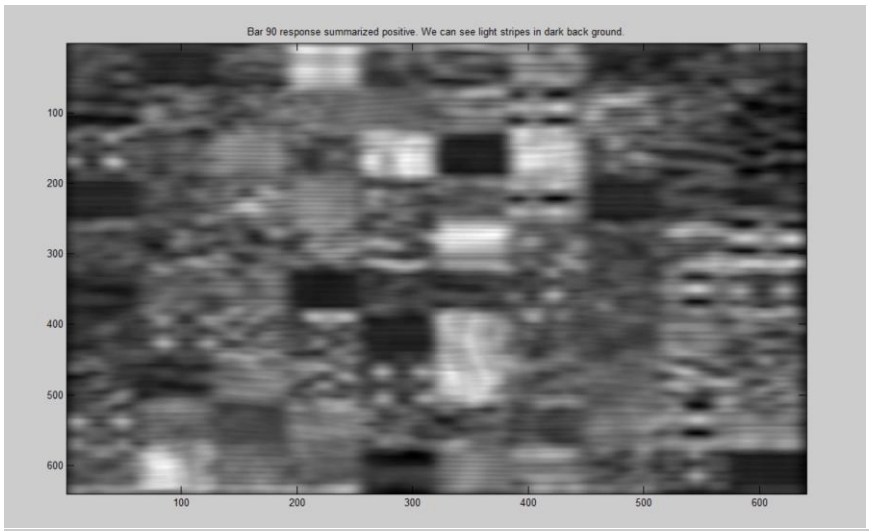

Negative rectification of the response of the image with the bar filter with zero rotation
100
200
300
400
500
600
100
200
300
400
500
600

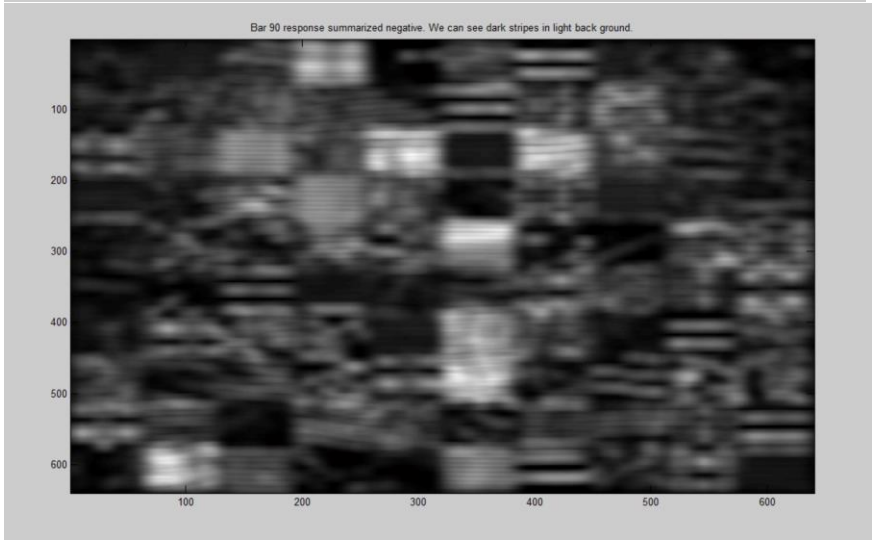

Bar 0 response summarized response. We can see light stripes in dark back ground.
100
200
300
400
500
600
100
200
300
400
500
600
Bar 0 response summarized negative. We can see dark stripes in light back ground.
100
200
300
400
500
600
100
200
300
400
500
600


Response of image with bar filter rotated at 90 degrees to find horizontal stripes
100
200
300
400
500
600
100
200
300
400
500
600


Positive rectification of the response of the image with the bar filter with 90 degrees rotation
100
200
300
400
500
600
100
200
300
400
500
600

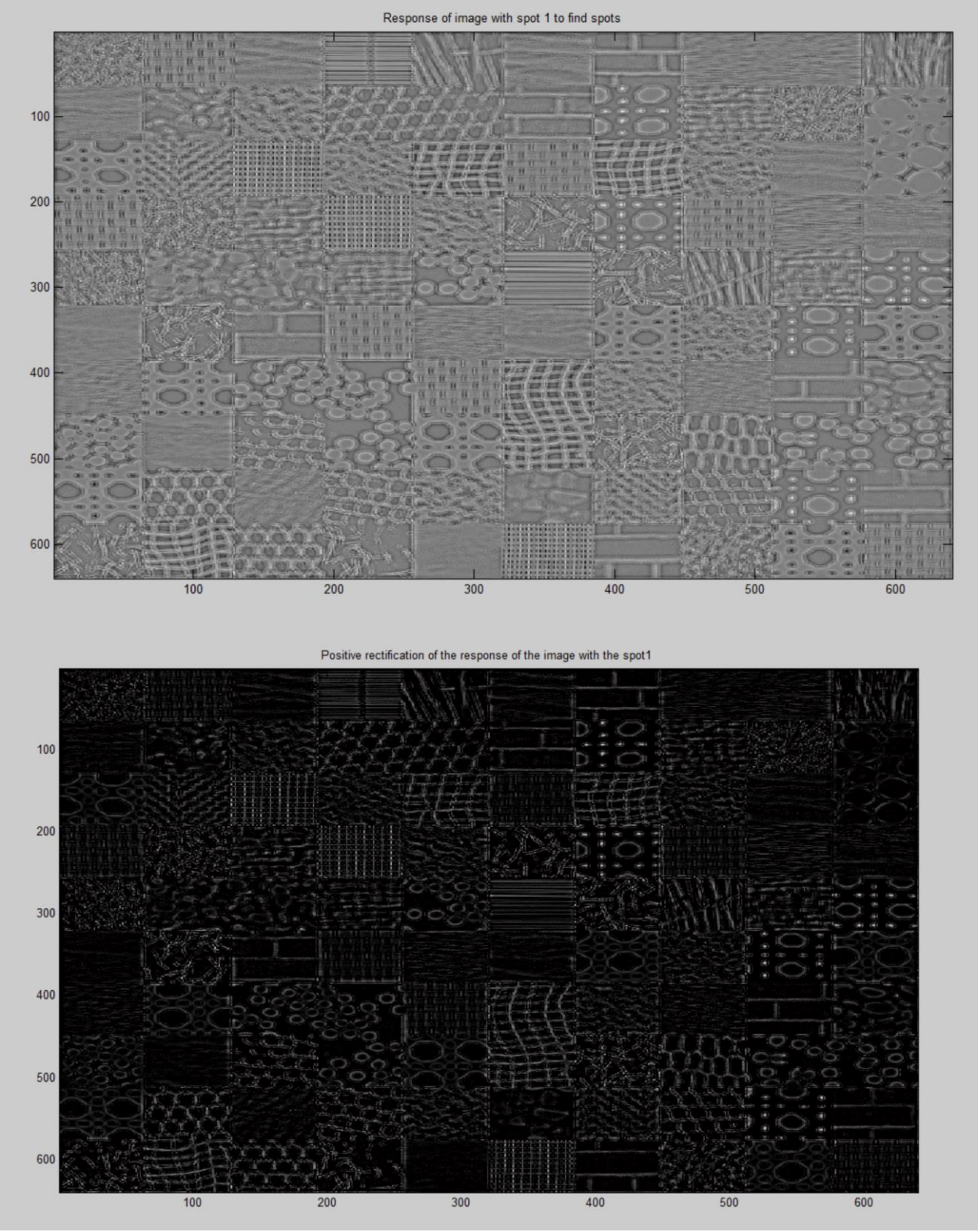

Negative rectification of the response of the image with the bar filter with 90 degrees rotation
100
200
300
400
500
600
100
200
300
400
500
600

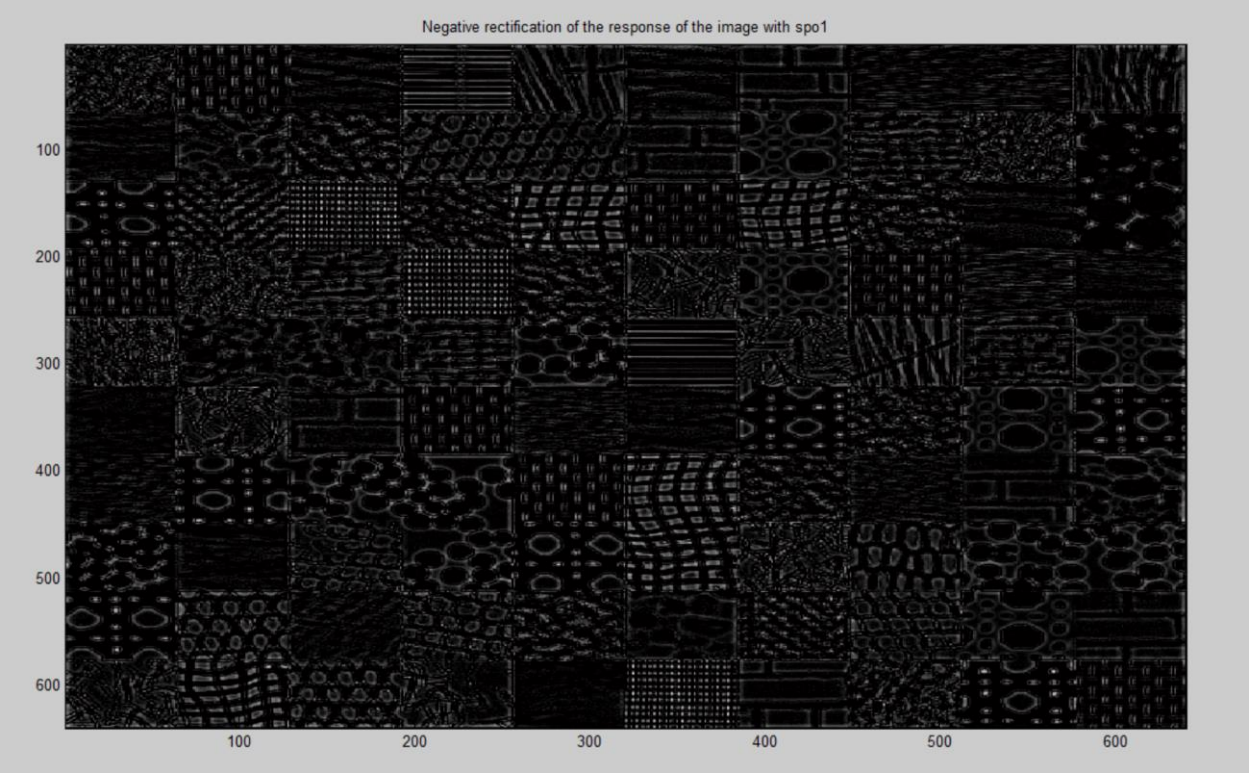


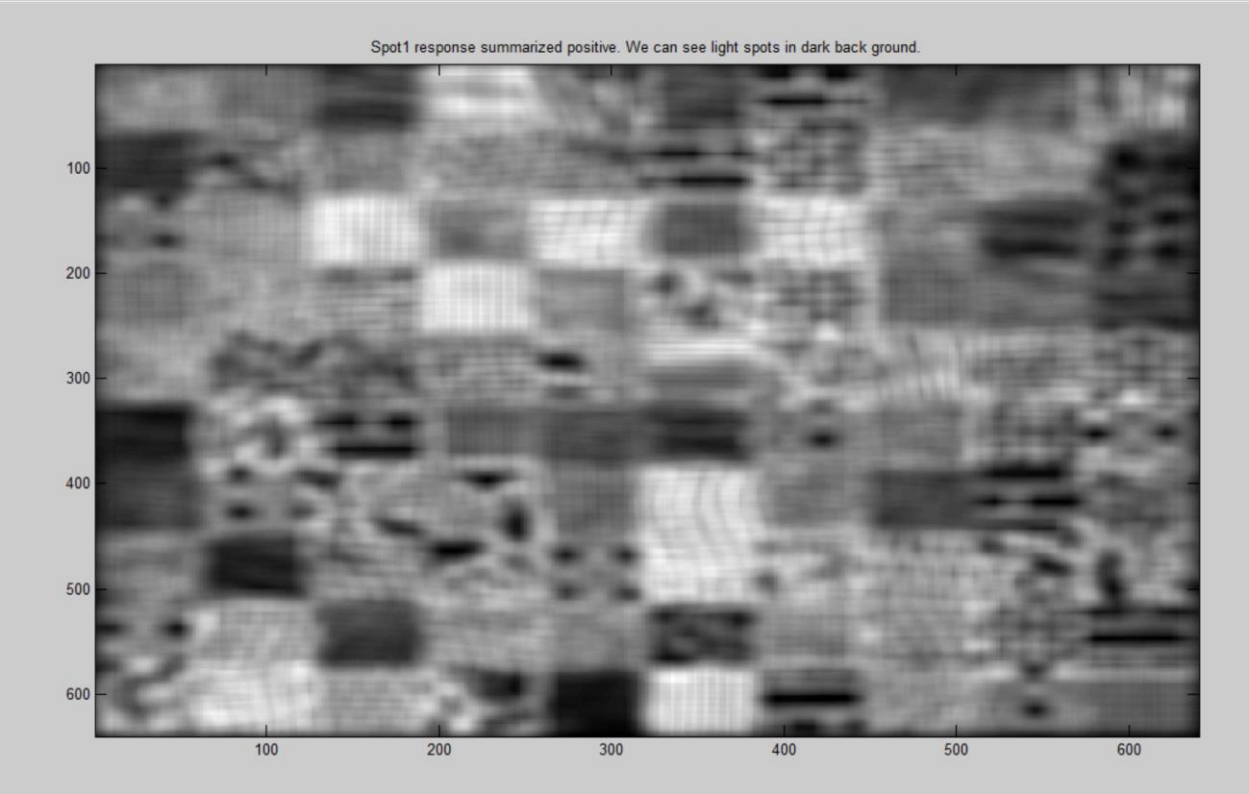


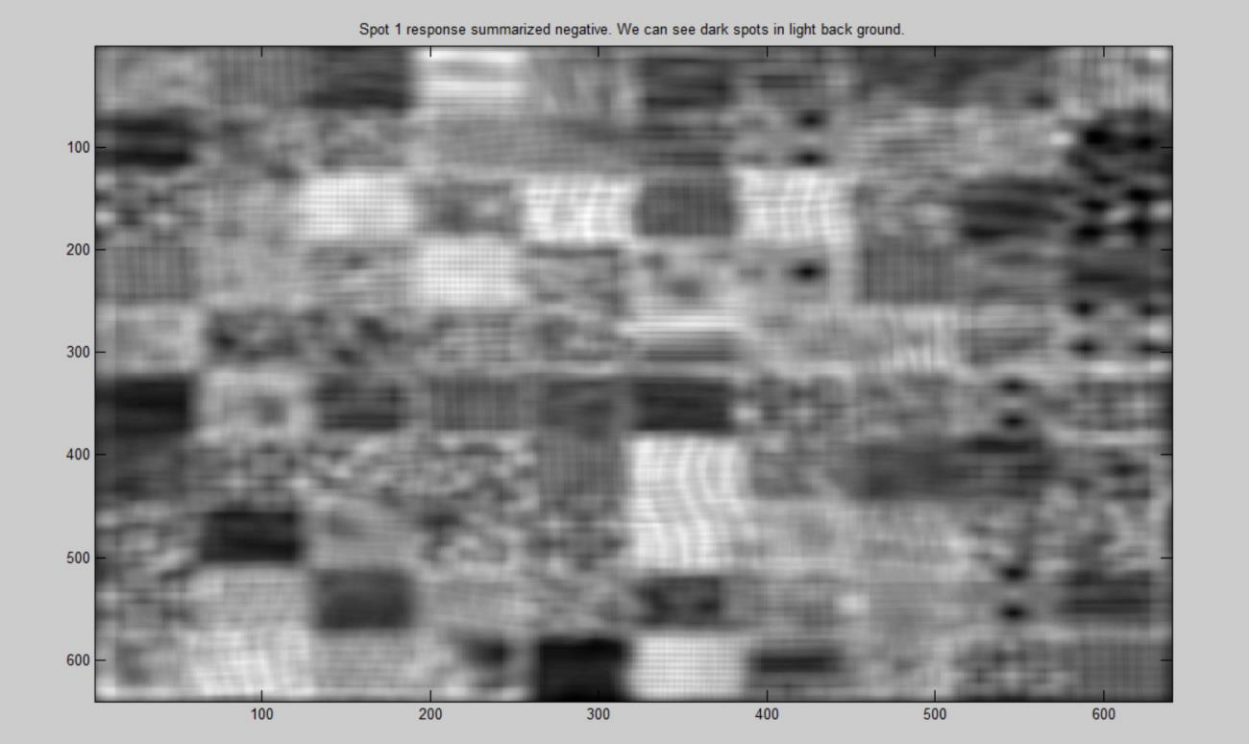
2)

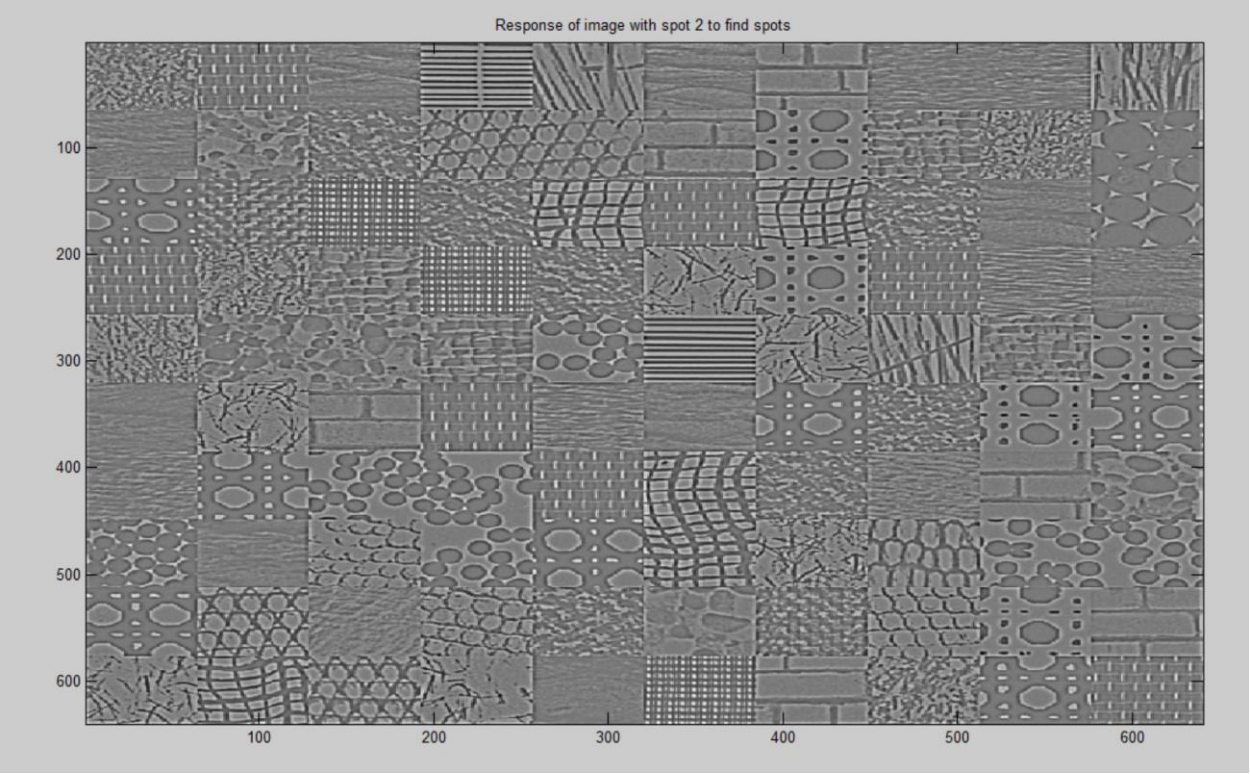
<u>In this test, we will show that the spot functions respond to spots, and bar function responds to oriented edges. Bar 0 and bar 90 have already been analysed in test 1. They are giving good horizontal and vertical stripes. Now lets test the two spot filters.</u>

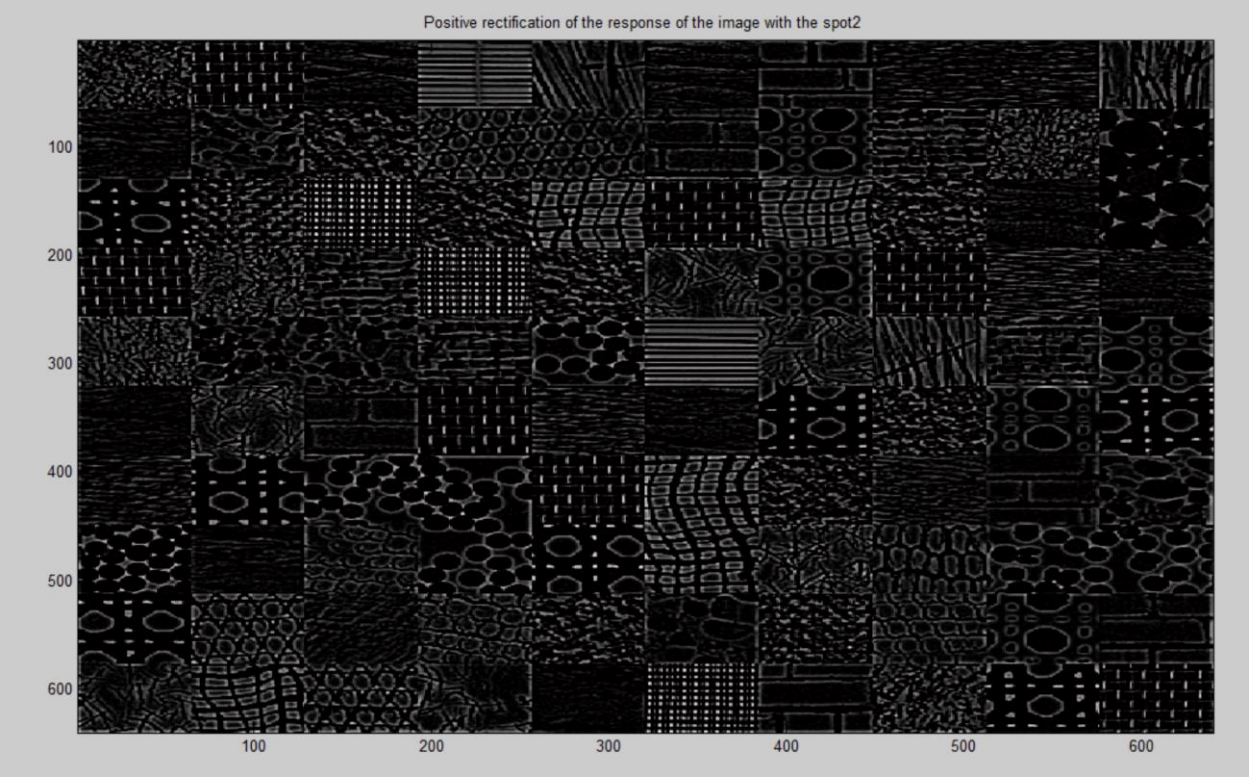

Response of image with spot 1 to find spots
100
200
300
400
500
600
100
200
300
400
500
600
Positive rectification of the response of the image with the spot1
100
200
300
400
500
600
100
200
300
400
500
600

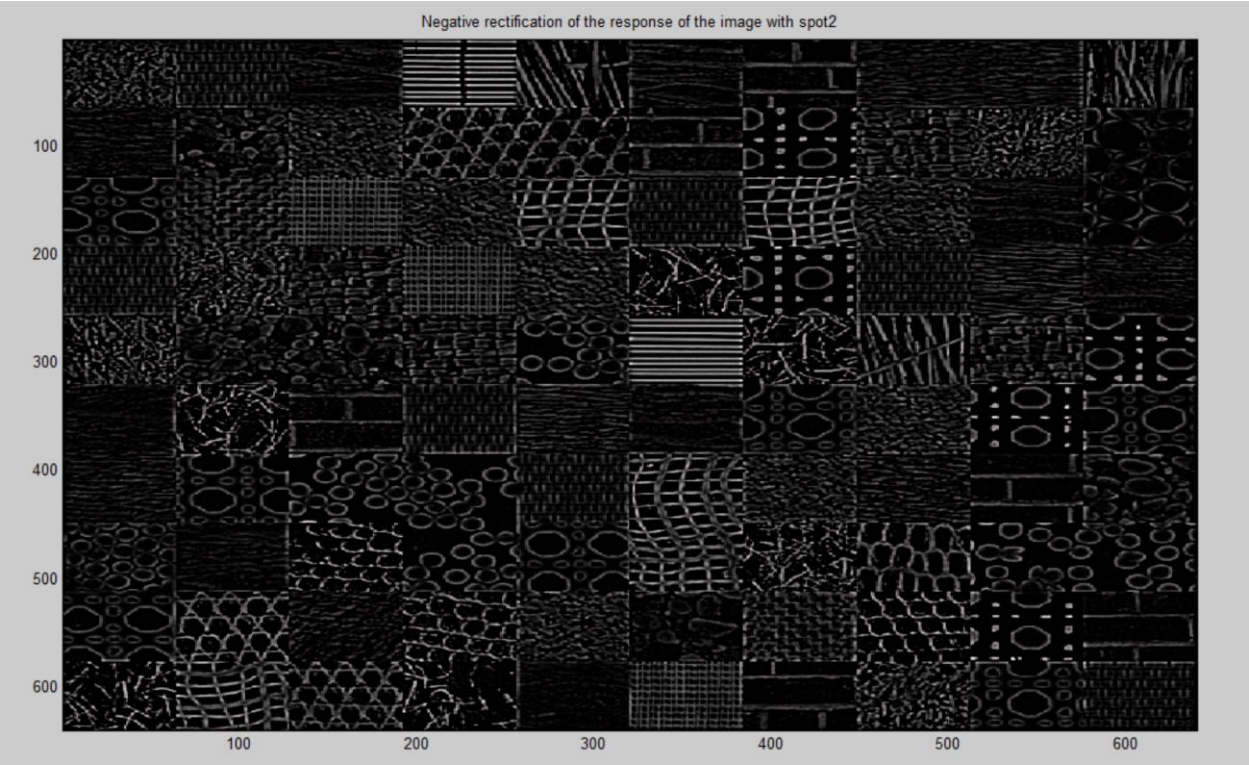

Negative rectification of the response of the image with spo1
100
200
300
400
500
600
100
200
300
400
500
600

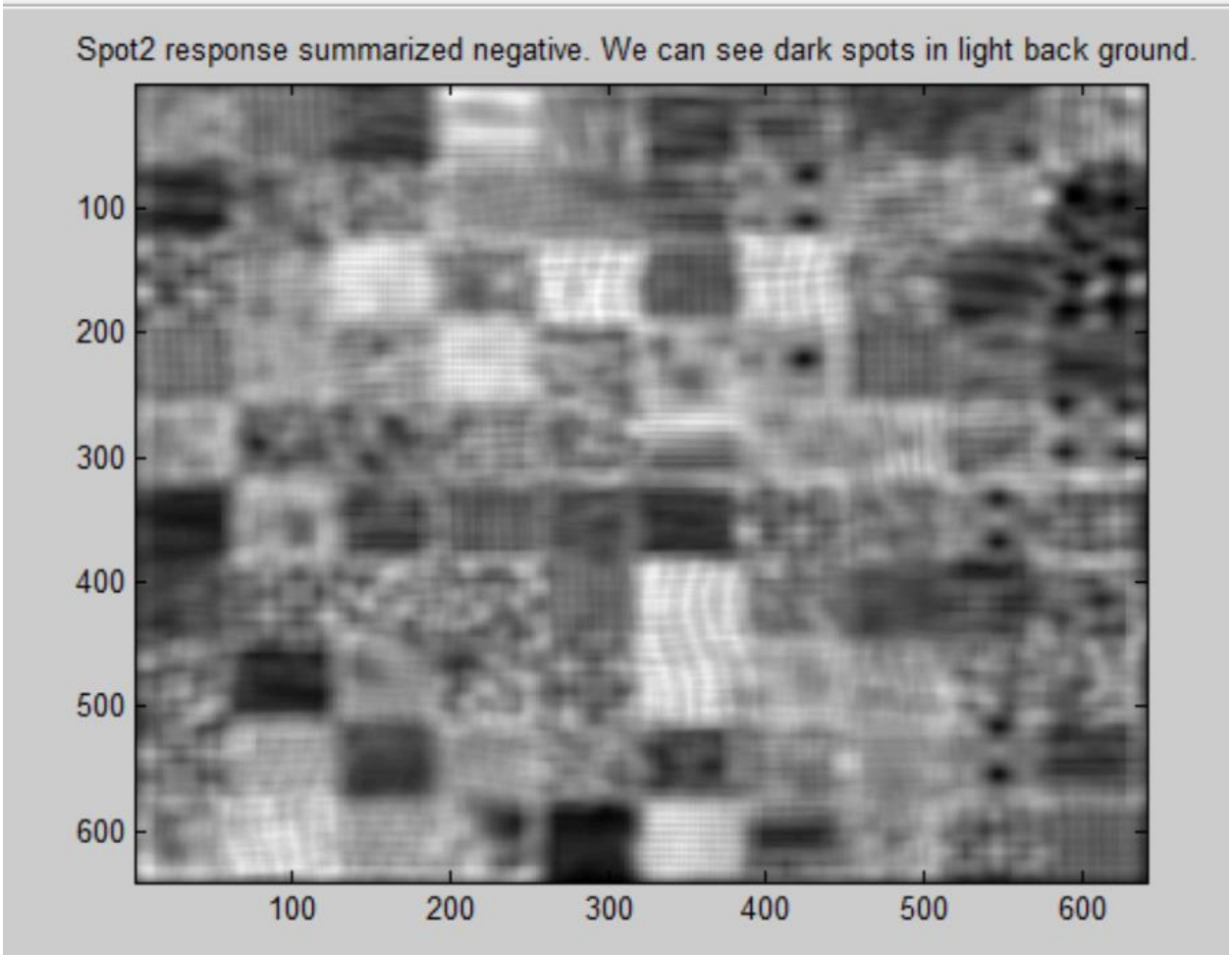

Spot1 response summarized positive. We can see light spots in dark back ground.
100
200
300
400
500
600
100
200
300
400
500
600

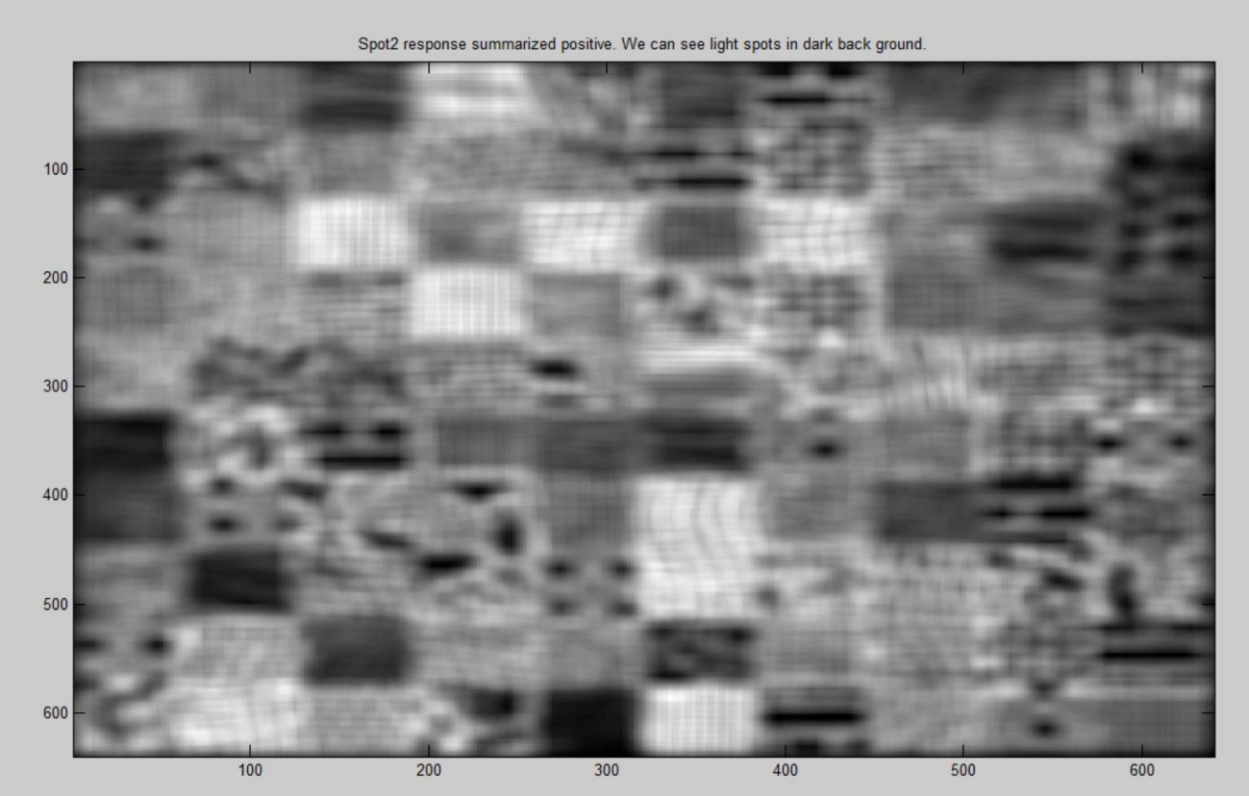

Spot 1 response summarized negative. We can see dark spots in light back ground.
100
200
300
400
500
600
100
200
300
400
500
600


Response of image with spot 2 to find spots
100
200
300
400
500
600
100
200
300
400
500
600


Positive rectification of the response of the image with the spot2
100
200
300
400
500
600
100
200
300
400
500
600


Negative rectification of the response of the image with spot2
100
200
300
400
500
600
100
200
300
400
500
600

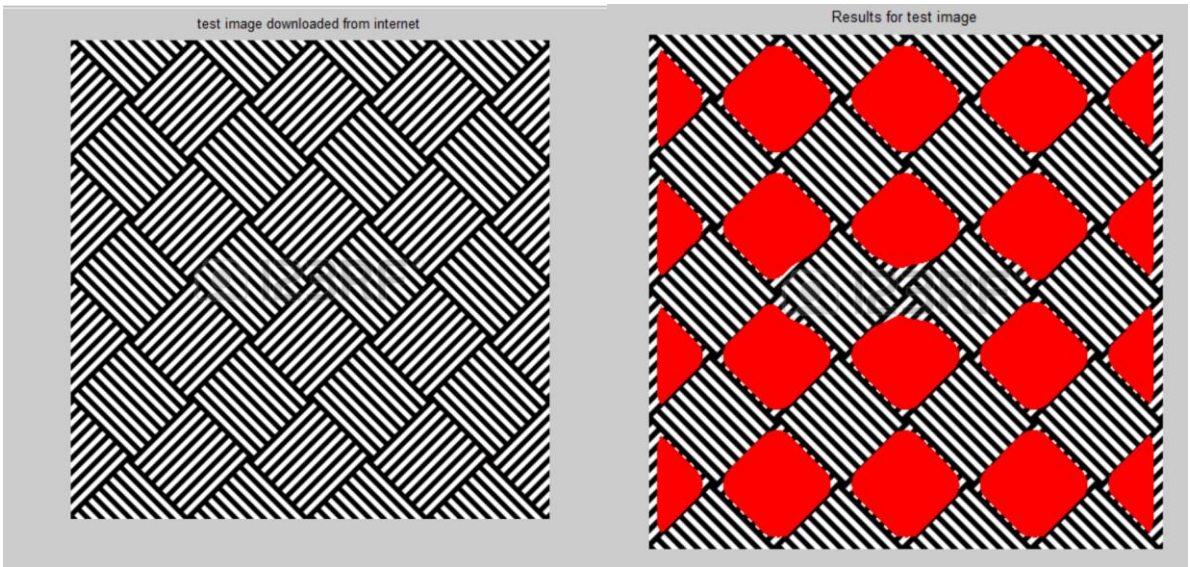

Spot2 response summarized negative. We can see dark spots in light back ground.
100
200
300
400
500
600
100
200
300
400
500
600



Thus, by above tests, I have tested my functions for Spot1, spot2, Bar0 and Bar 90.

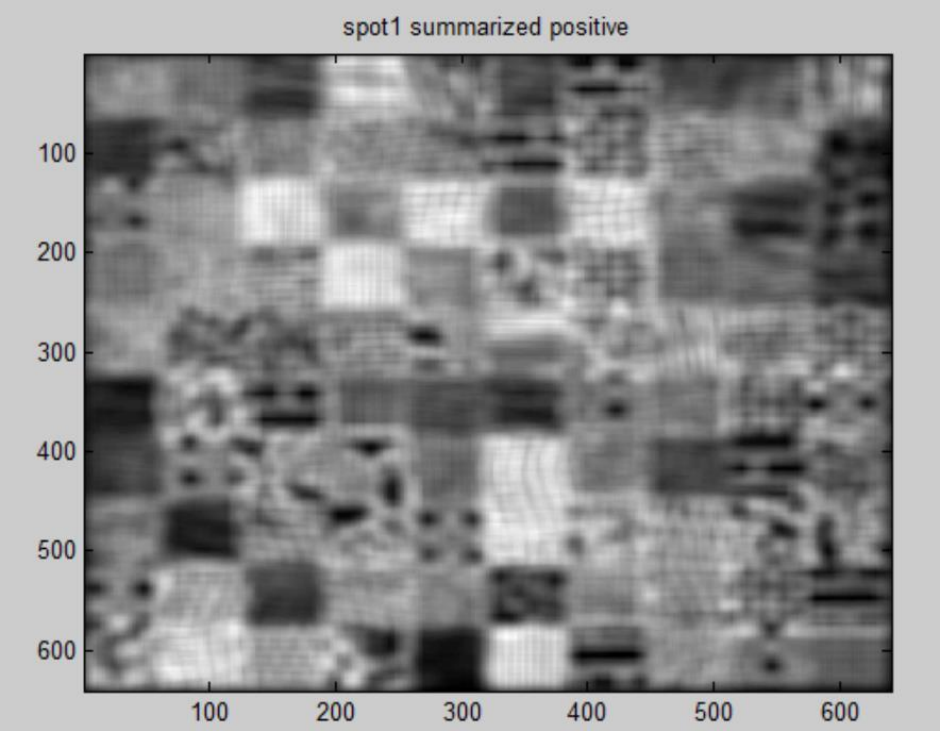
Testing CS5320_k_means_texture

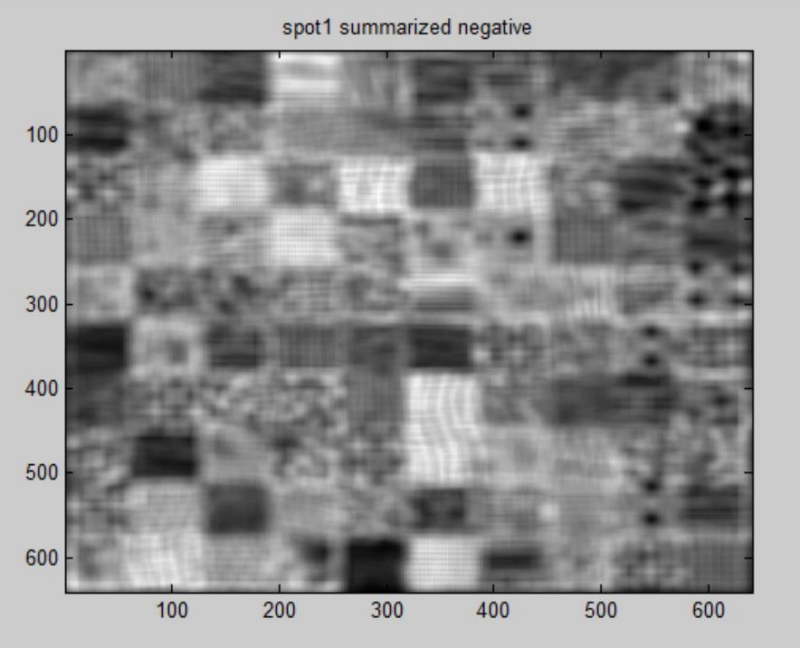
To test this function, I downloaded a very basic image rom internet and got the results as follows. These are for k = 2

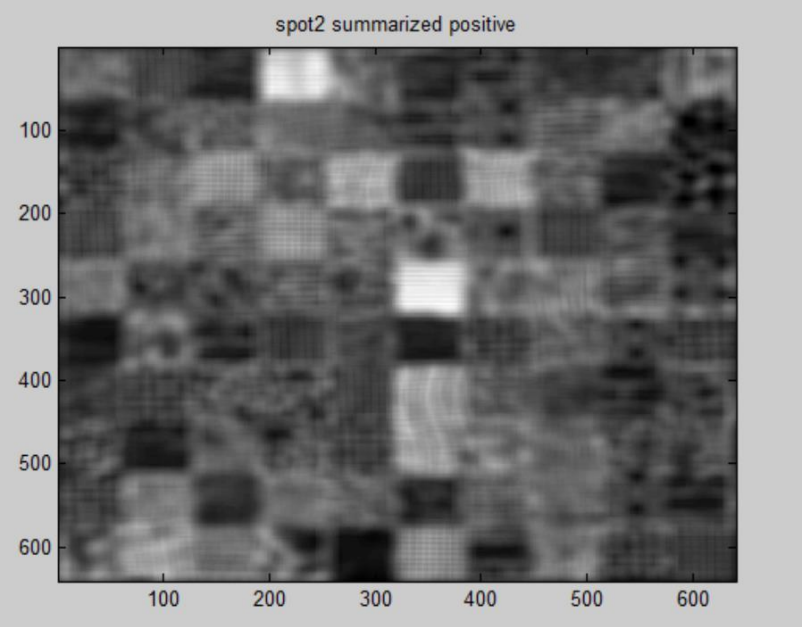


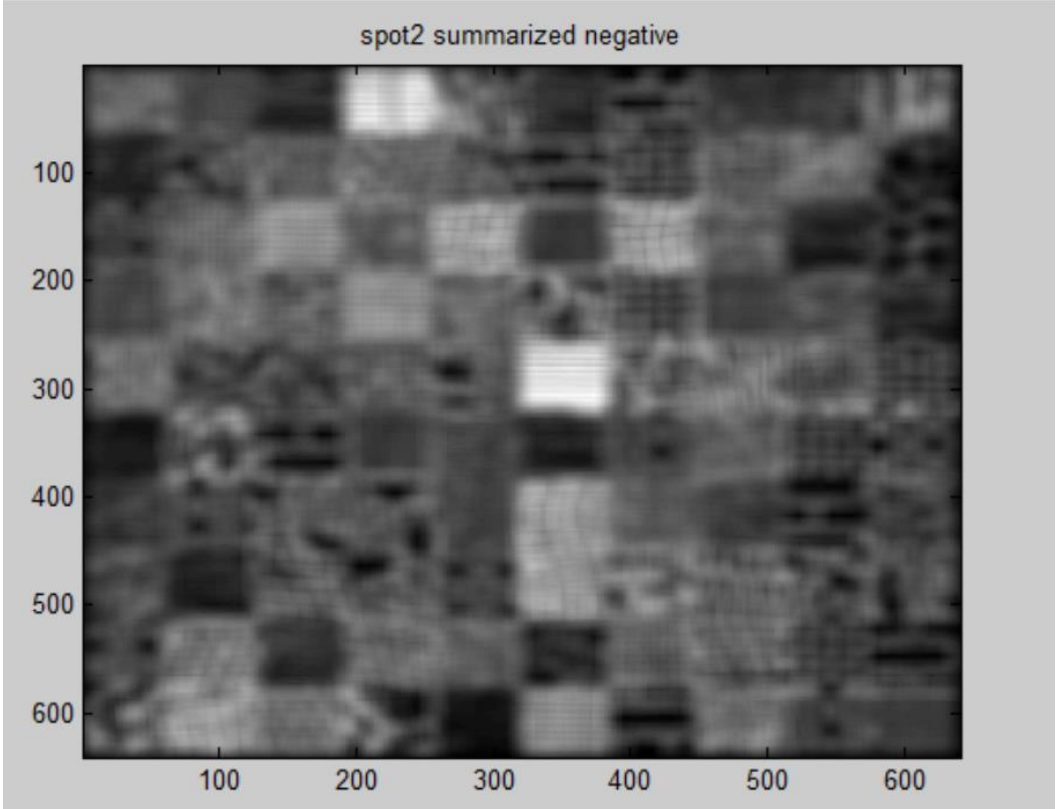
Section 4: Data:

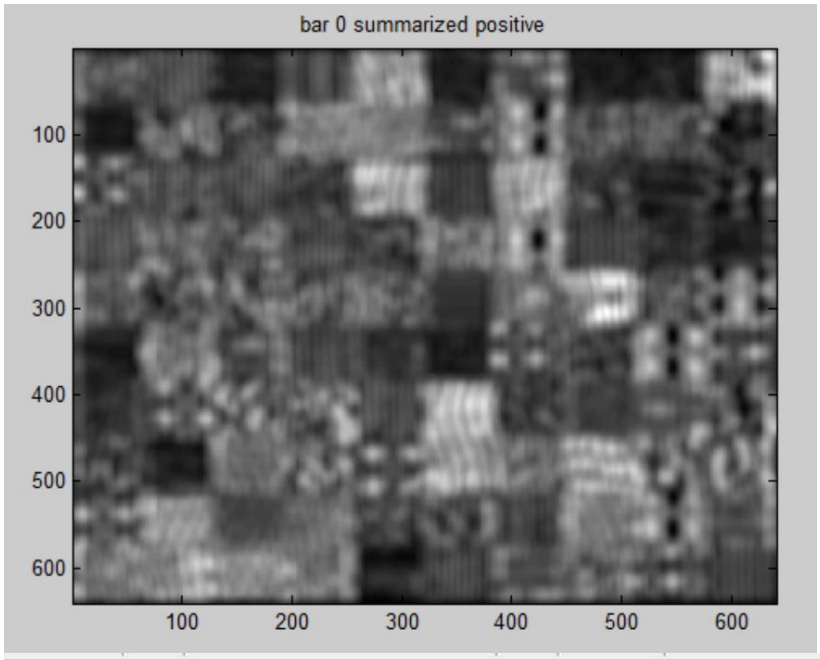
The following figures contain results for my CS5320_texture_params function for Gaussian of size 22X22 and sigma = 7.

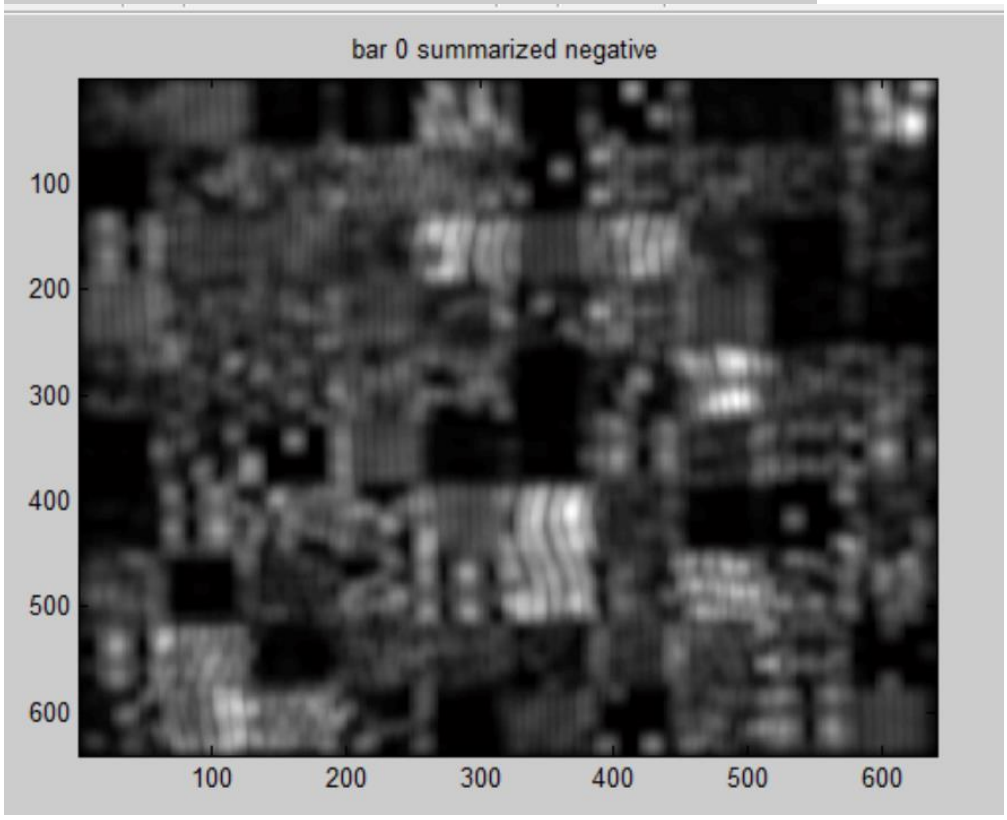


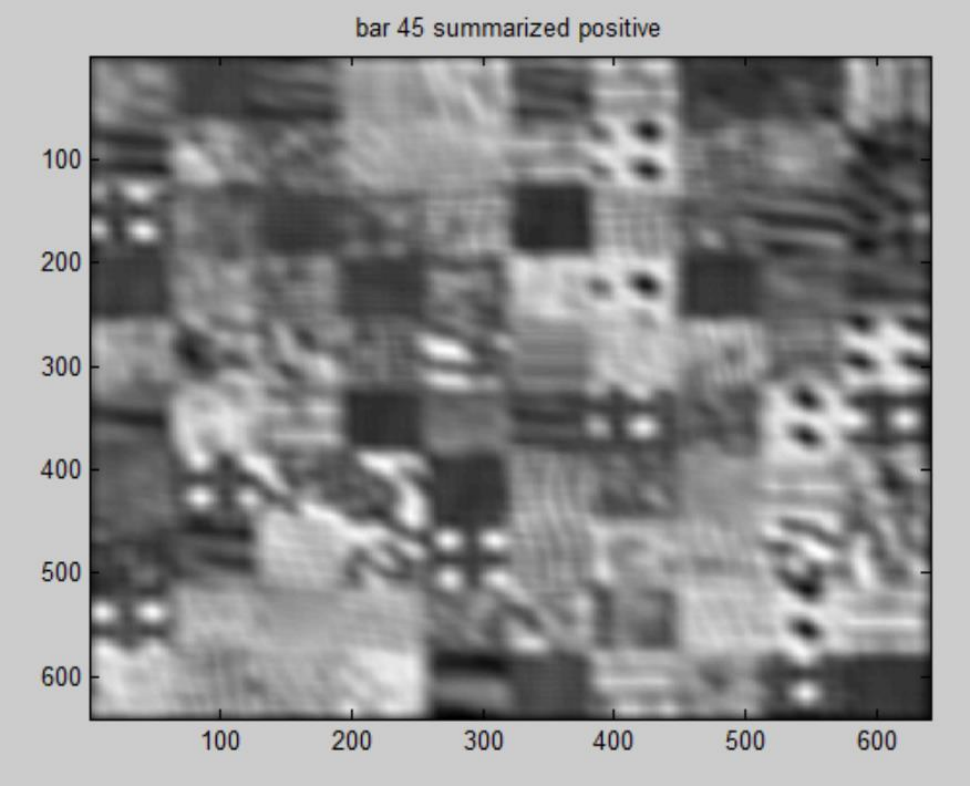


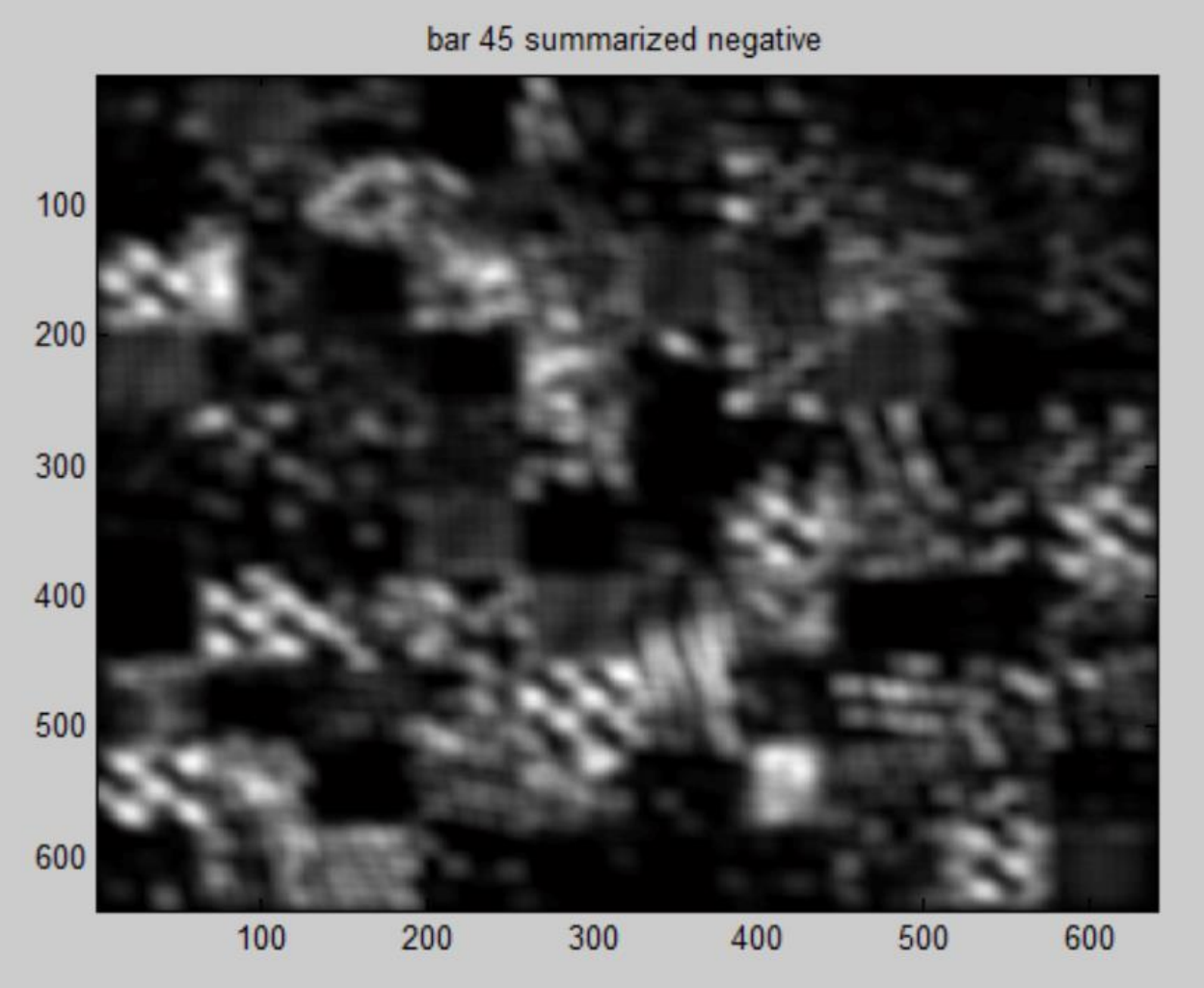

spot2 summarized positive
100
200
300
400
500
600
100
200
300
400
500
600

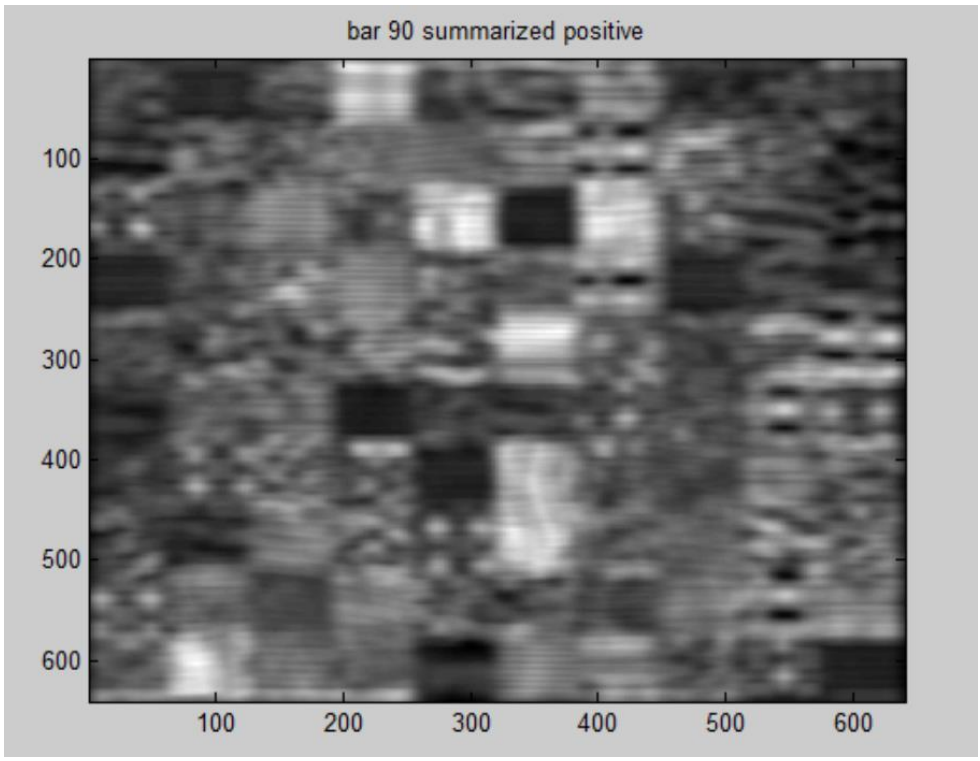

spot2 summarized negative
100
200
300
400
500
600
100
200
300
400
500
600

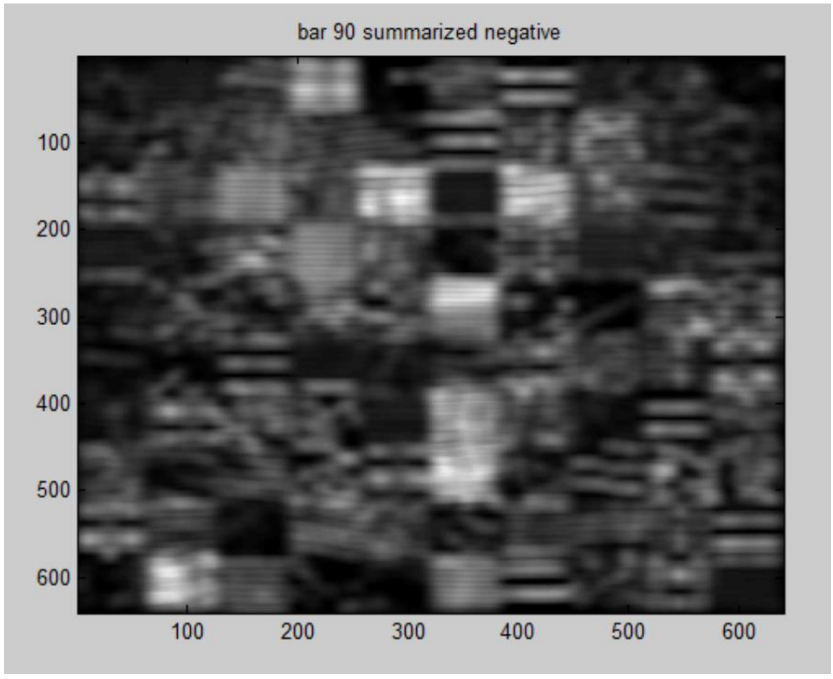

bar 0 summarized positive
100
200
300
400
500
600
100
200
300
400
500
600

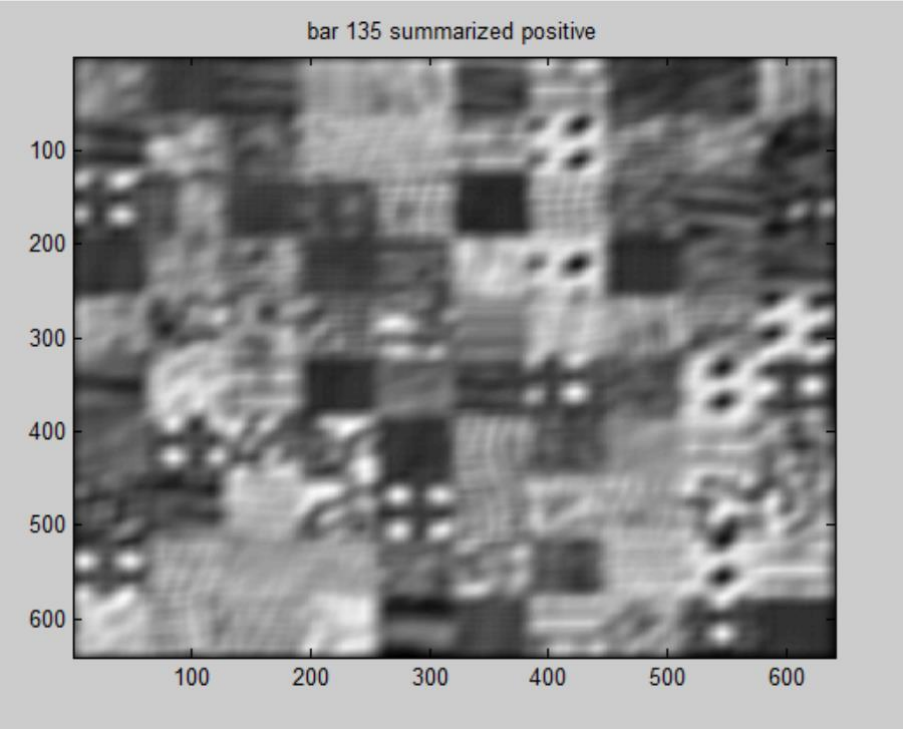

bar 0 summarized negative
100
200
300
400
500
600
100
200
300
400
500
600

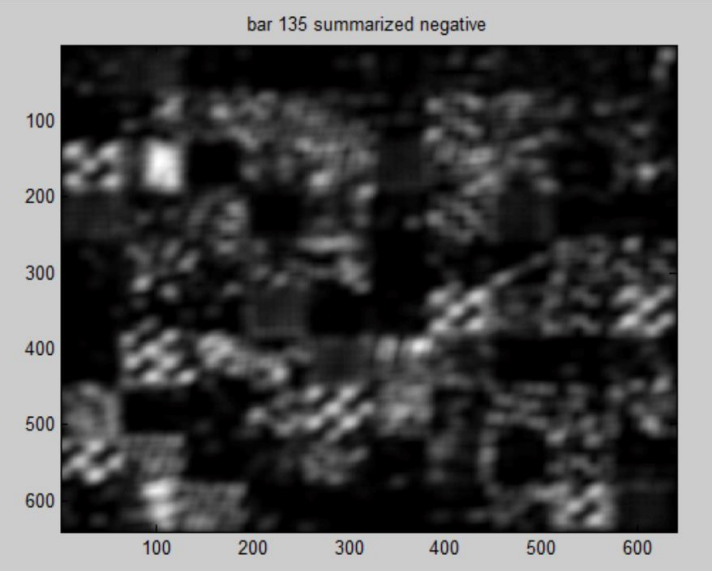

bar 45 summarized positive
100
200
300
400
500
600
100
200
300
400
500
600

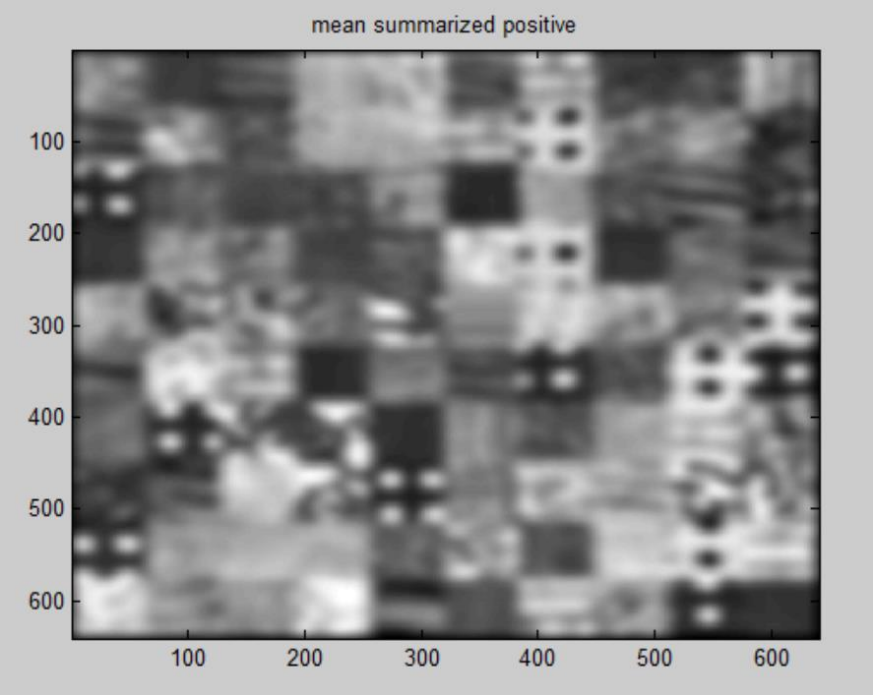

bar 45 summarized negative
100
200
300
400
500
600
100
200
300
400
500
600

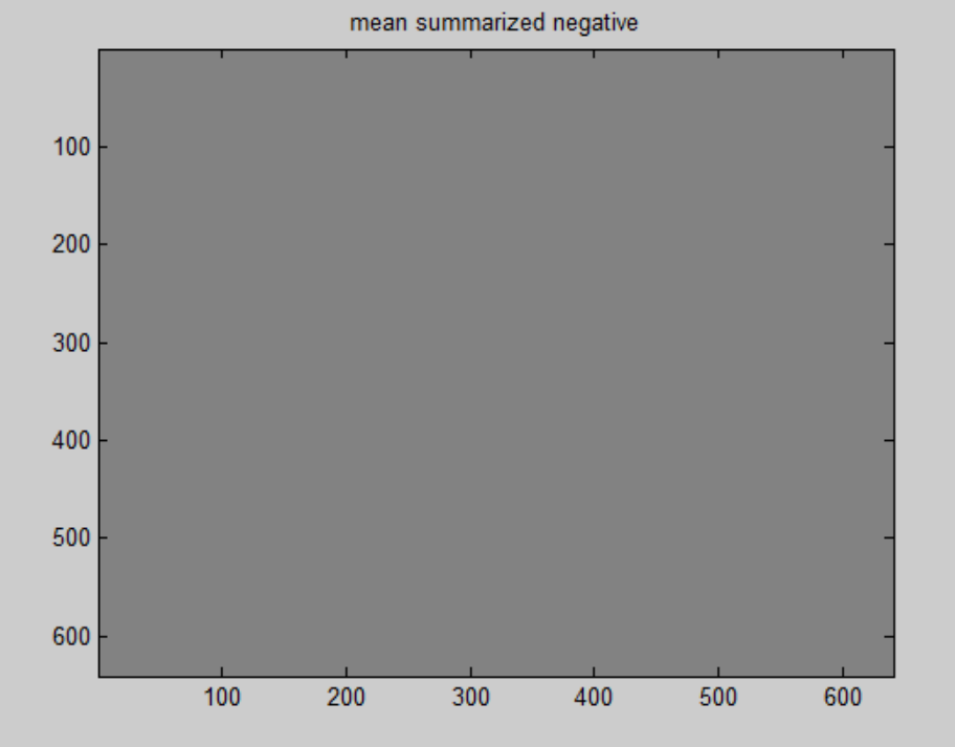

bar 90 summarized positive
100
200
300
400
500
600
100
200
300
400
500
600

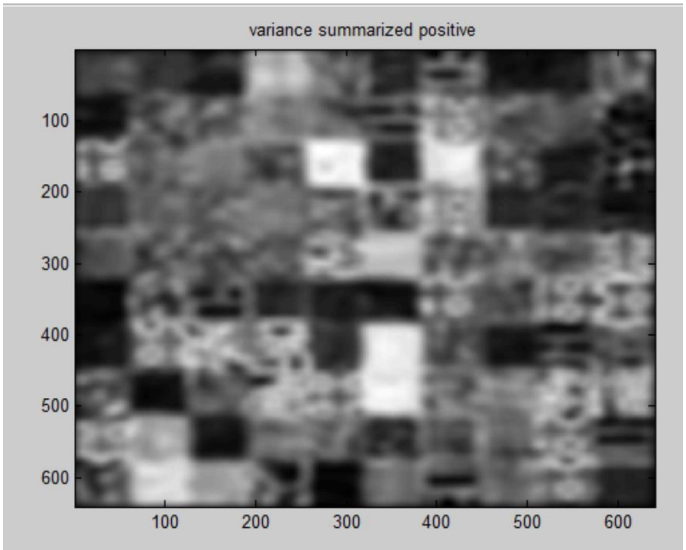

bar 90 summarized negative
100
200
300
400
500
600
100
200
300
400
500
600

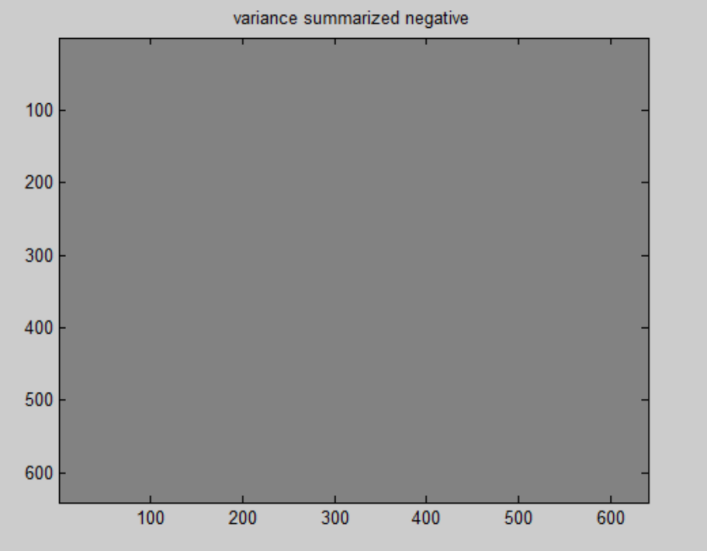

bar 135 summarized positive
100
200
300
400
500
600
100
200
300
400
500
600


bar 135 summarized negative
100
200
300
400
500
600
100
200
300
400
500
600

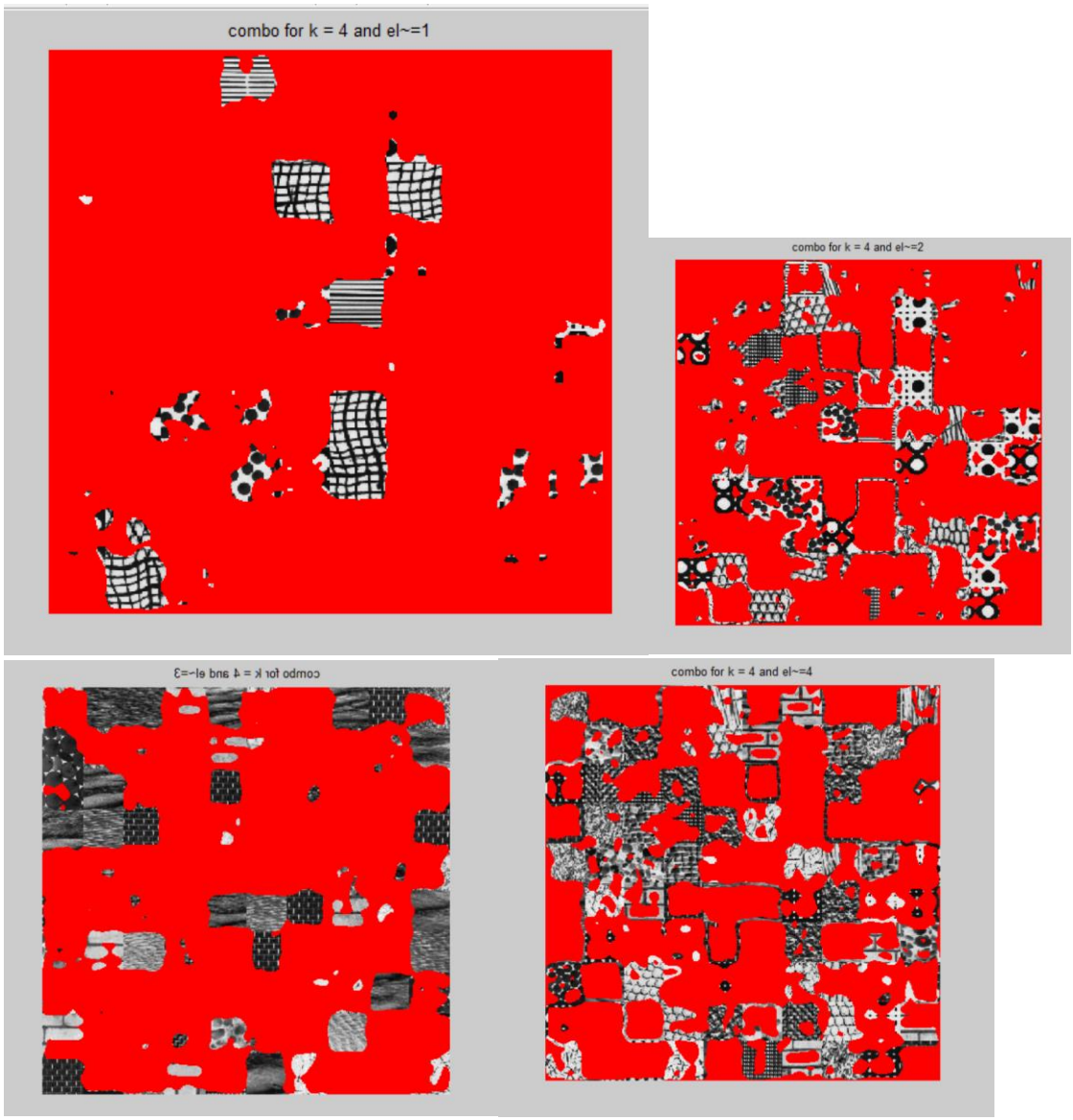

mean summarized positive
100
200
300
400
500
600
100
200
300
400
500
600

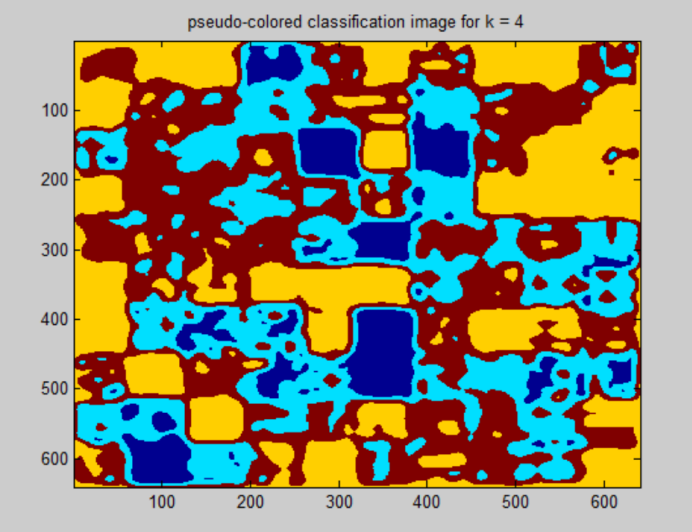

mean summarized negative
100
200
300
400
500
600
100
200
300
400
500
600

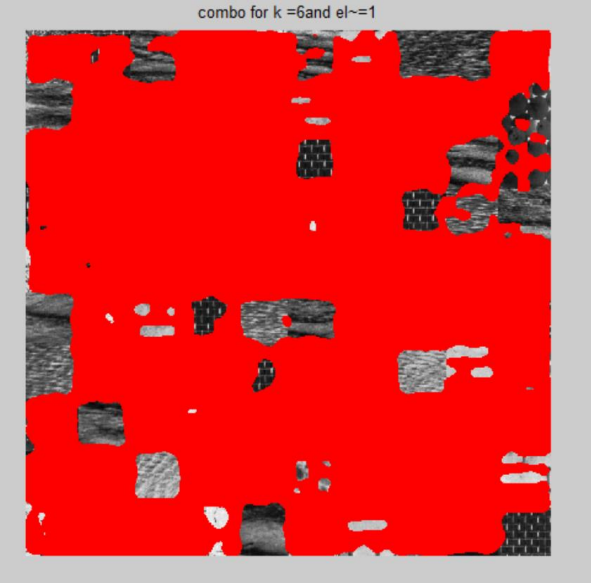

variance summarized positive
100
200
300
400
500
600
100
200
300
400
500
600

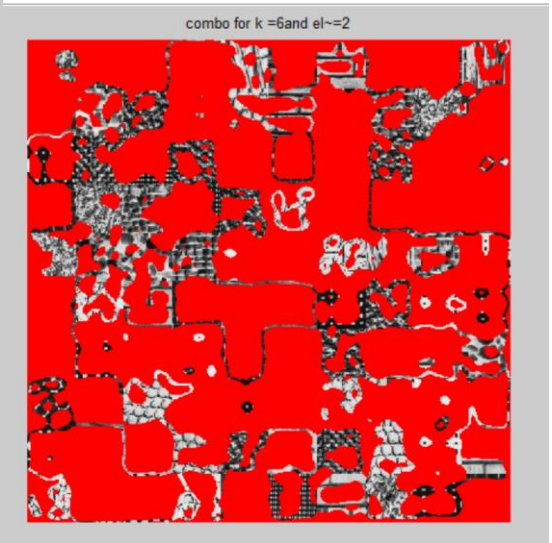

variance summarized negative
100
200
300
400
500
600
100
200
300
400
500
600

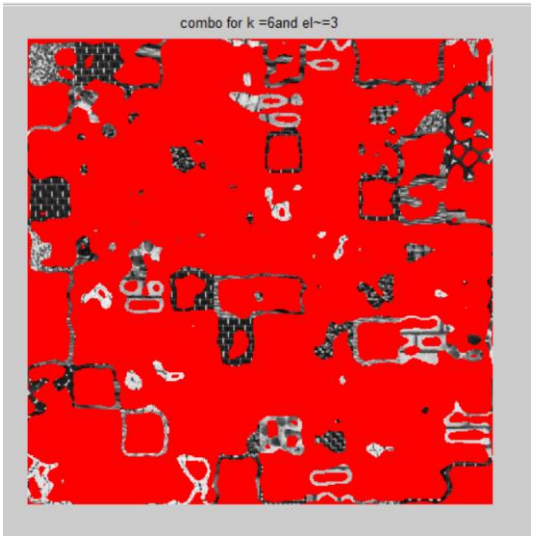
NOW, We will draw our results for various values of k in the following figures:

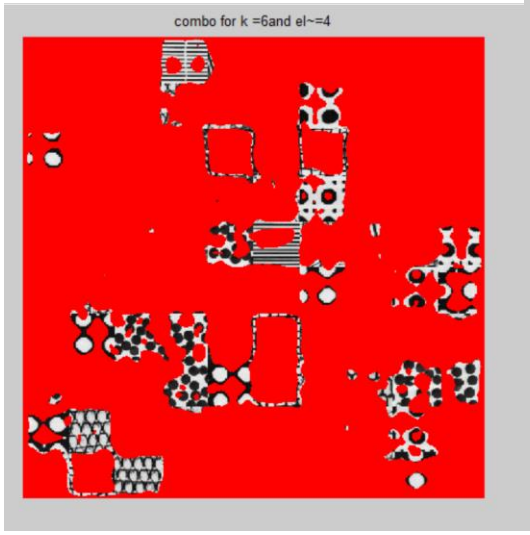


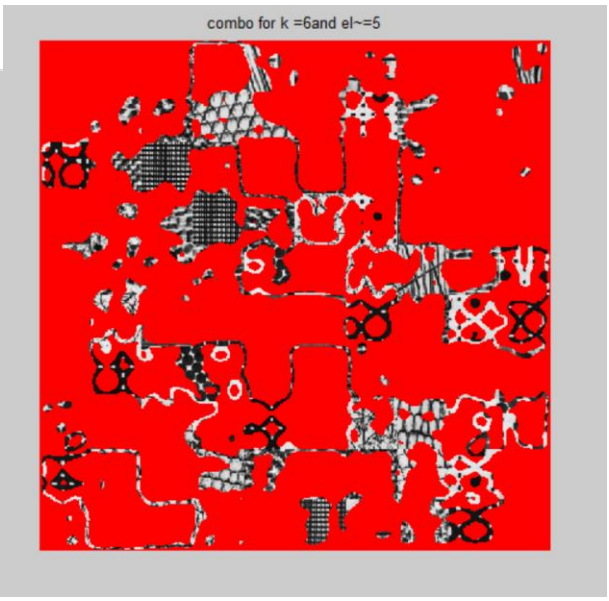

pseudo-colored classification image for k = 4
100
200
300
400
500
600
100
200
300
400
500
600

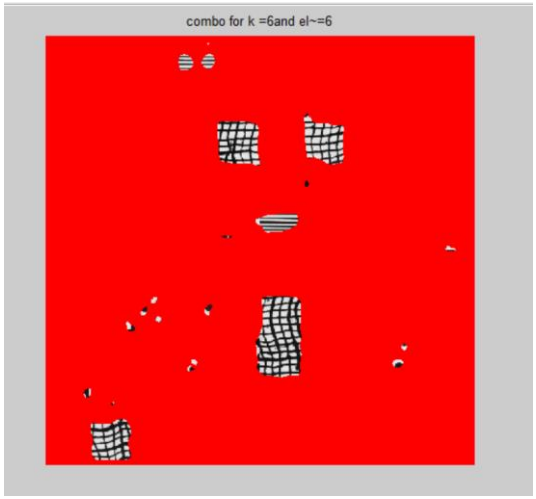

combo for k =6and el~=1

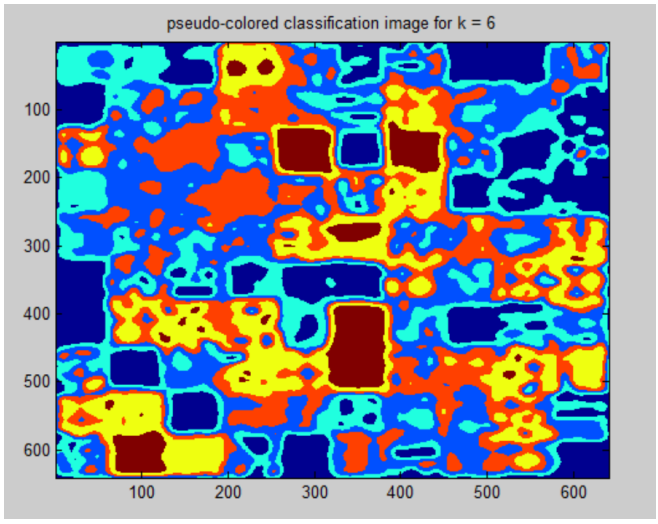

combo for k =6and el~=2

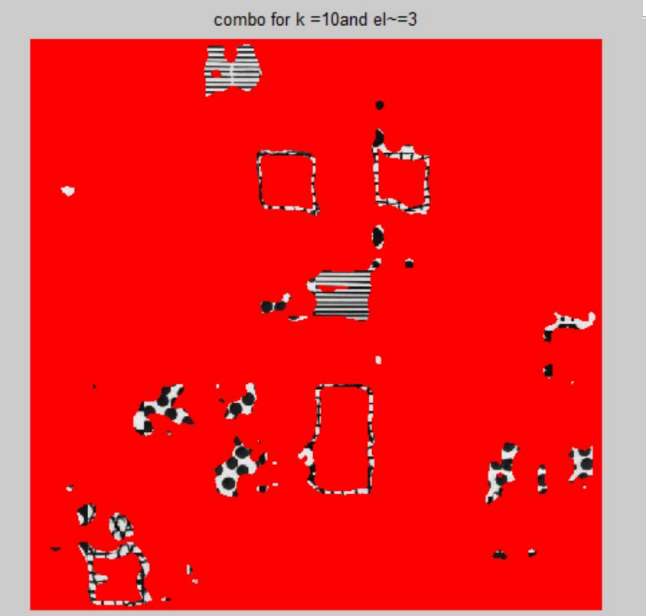

combo for k =6and el~=3

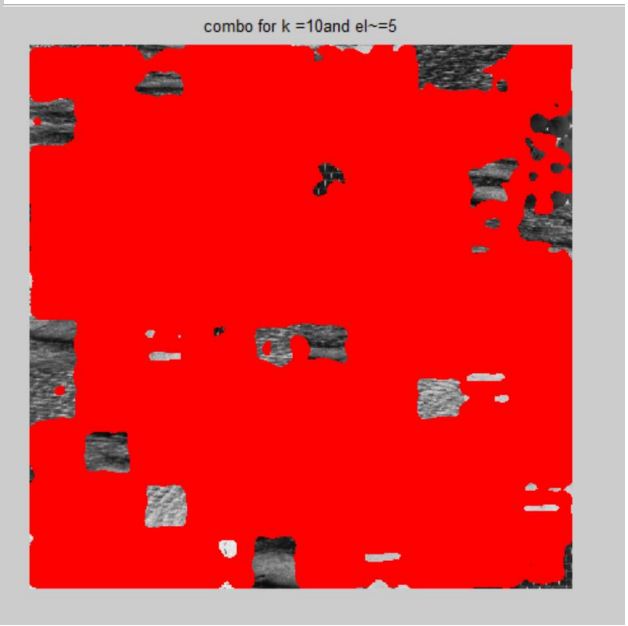

combo for k =6and el~=4


combo for k =6and el~=5

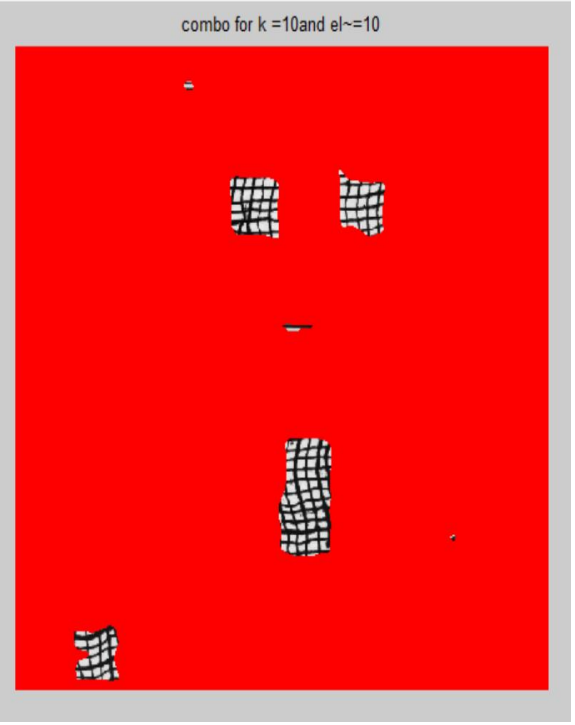

combo for k =6and el~=6

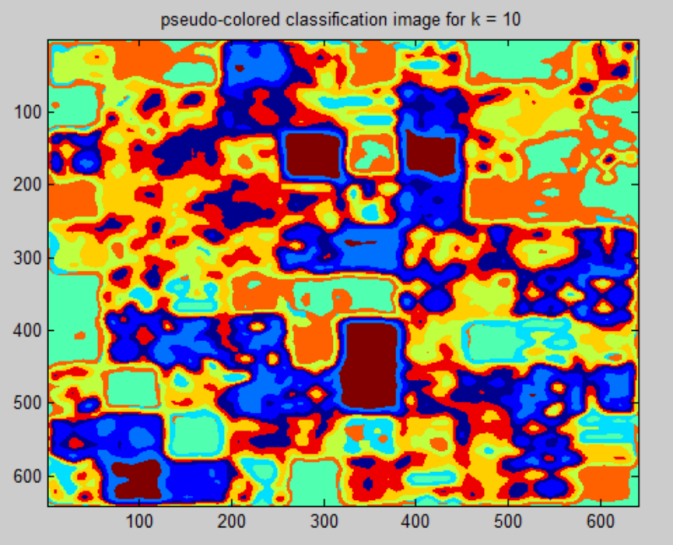

pseudo-colored classification image for k = 6
100
200
300
400
500
600
100
200
300
400
500
600


combo for k =10and el~=3


combo for k =10and el~=5

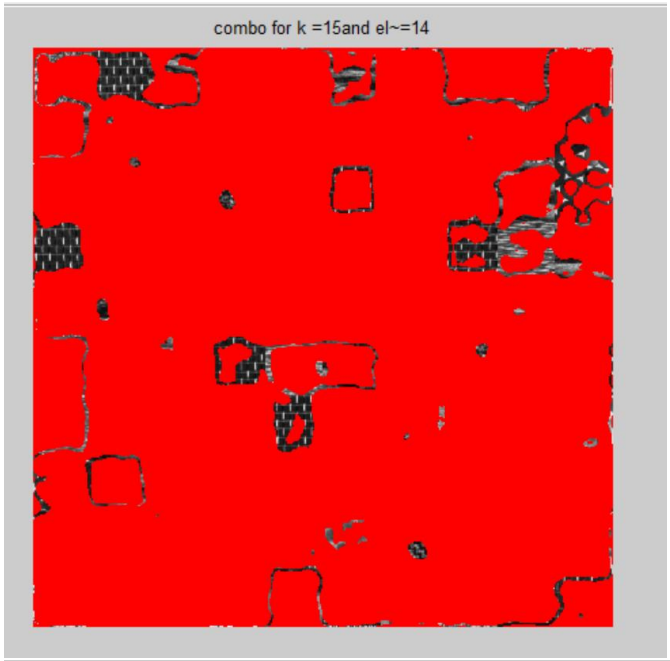

combo for k =10and el~=8

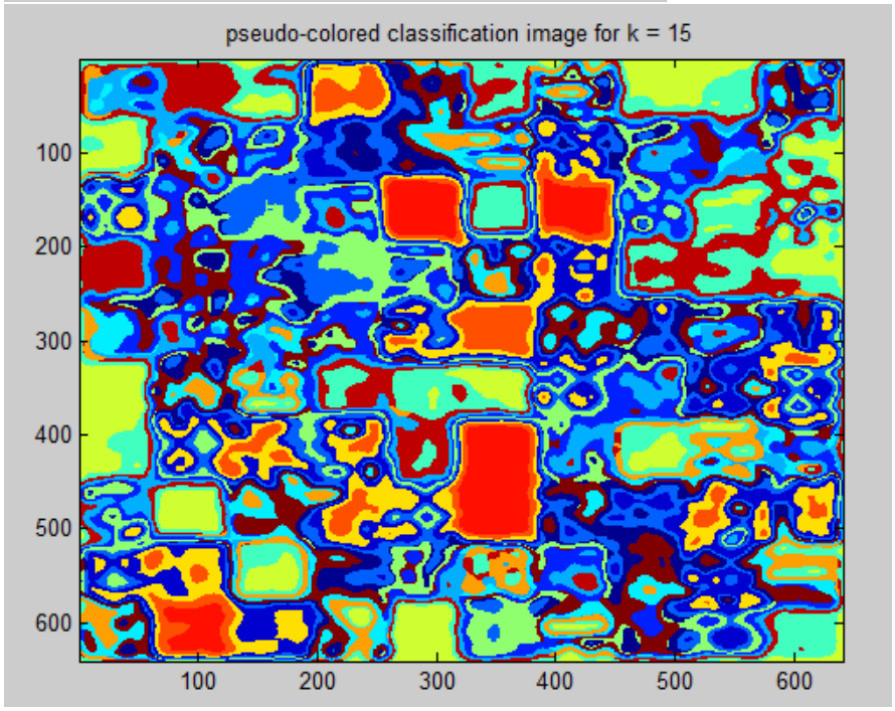

combo for k =10and el~=10

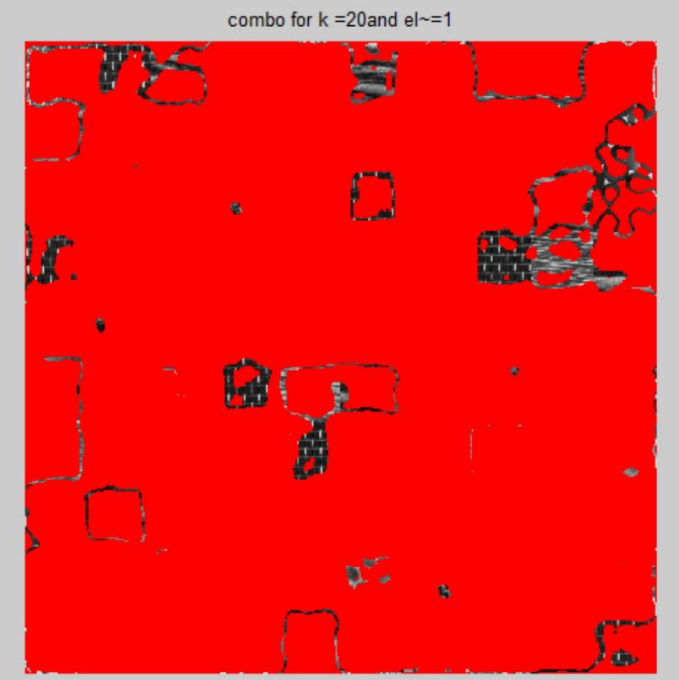

pseudo-colored classification image for k = 10
100
200
300
400
500
600
100
200
300
400
500
600


combo for k =15and el~=9


combo for k =15and el~=13

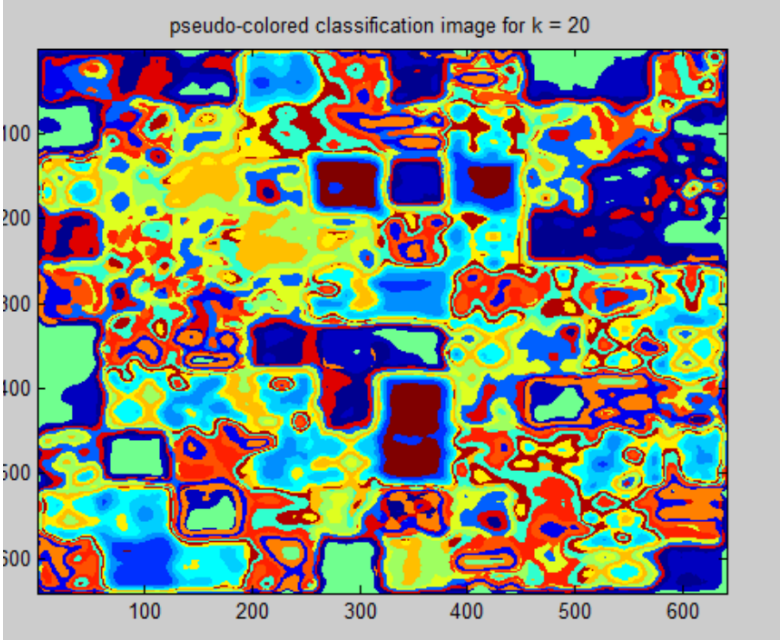

combo for k =15and el~=14


pseudo-colored classification image for k = 15
100
200
300
400
500
600
100
200
300
400
500
600


combo for k =20and el~=1


combo for k =20and el~=10


combo for k =20and el~=20



Section 5: Analysis:

Here, we will be doing analyses for various values of k (4,6,10,15,20) by computing statistics on the number of good texture classification vs k. The figures from the data section are referred for this purpose.

K = 4;

I am able to properly get only the following patch properly:

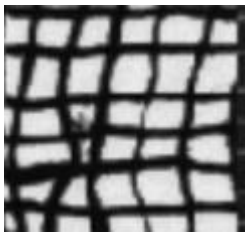


Figure 3 Patch1

Other patches are not coming properly. This is because the number of clusters i.e. 4 is very small.

Number of squares similar to patch1 = 5

Number of good texture classification for patch1 as computed by me: 5

Percentage of patches having same class(Number of patches correctly found/Number of those patches present in the image) = 100%

K = 6;

Again I am getting patch 1 good.

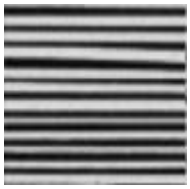
Number of squares similar to patch1 in the image = 5

Number of good texture classification for patch1 as computed by me: 5

Percentage of patches having same (Number of patches correctly found/Number of those patches present in the image)= 100%

However, the number of pixels in each square that have same label have decreased as compared to k = 4

The area of undesired patches detected have decreased. For example, with k = 4, the following patch was coming unnecessarily. Now it coming for only very small area.



K = 10;

Again I am getting patch 1 good.

Number of squares similar to patch1 in the image = 5

Number of good texture classification for patch1 as computed by me: 5

Percentage of patches having same class(Number of patches correctly found/Number of those patches present in the image) = 100%

In addition to patch 1, I am now also getting following patch (although it is not perfect):

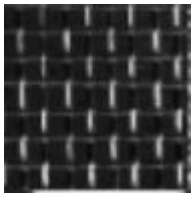


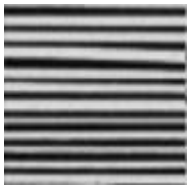
Figure 4Patch 2

Number of squares similar to patch2 in the image= 7

Number of good texture classification for patch2 as computed by me: 6

Percentage of patches having same class = 85.7

For patch1, the area of undesired patches detected have decreased. For example, with k = 6, the following patch was coming unnecessarily. Now it is completely gone.



K = 15;

Again I am getting patch 1 good.

Number of squares similar to patch1 in the image = 5

Number of good texture classification for patch1 as computed by me: 5

Percentage of patches having same class = 100%

The percentage of patch 2 correctly recognized has decreased

Number of squares similar to patch2 in the image= 7

Number of good texture classification for patch2 as computed by me: 5

Percentage of patches having same class (Number of patches correctly found/Number of those patches present in the image) = 71%

K = 20;

The percentage of patch 1 correctly recognized has decreased as follows:

Number of squares similar to patch1 in the image= 5

Number of good texture classification for patch1 as computed by me: 4

Percentage of patches having same class = 80%

Also,

However, the number of pixels in each square that have same label have decreased significantly.

For k = 20, I am unable to detect patch 2 well.

Section 6: Interpretation:

The following are my observations:

- K = 10 seems to me as the optimal solution since I am able to determine well patch 1 and patch 2 (see figure 3 and 4 to see patch 1 and 2). Second reason for k = 10 being good is that the amount of undesired pattern recognition is least.
- As the k is in increased, the percentage of patches having same class (Number of patches correctly found/Number of those patches present in the image) first increases till 10 and then decreases.
- The max percentage of pixels in each square that have the same label seems to be decreasing with increase in k. Although I have not calculated the exact percentage, but by looking at by figures from combos, I am easily see that.
- Got best results for Gaussain size 22 and sigma 7

Section 7: Critique:

The experiment could be improved by following ways:

- Computing the decrease in the number of pixels in each square that have same label with the increase in k.
- Playing more with a,b,c, d for orientation Gaussian function

- **Section 8: log**
  20+ hours total