# Assignment A7: Texture

Clinton Fernandes

u1016390

03/23/2016

A7

CS 6320

## 1. Introduction

The purpose of this assignment is to classify textures on a mixed texture image. Texture is important, because it appears to be a very strong cue to an object's identity. For this, we will have to implement the Algorithm 6.3 from the text. Matlab functions will have to be developed for this, including the functions for spot and bar filters, texture parameters and a function for the k means clustering.

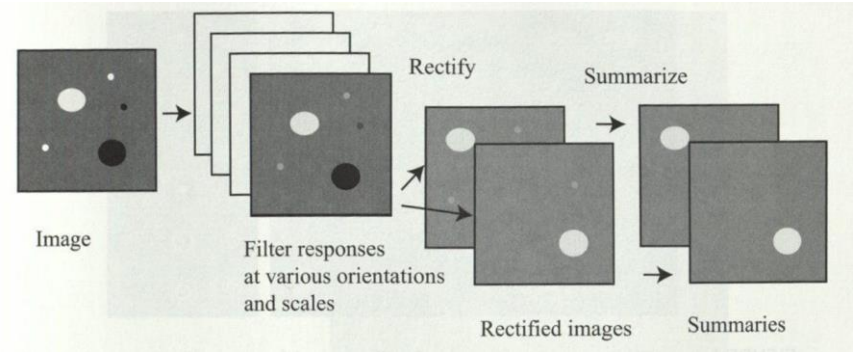
One question that can be asked is how does the number of good texture regions detected changes as we vary the number of clusters (input to kmeans function).

## 2. Method

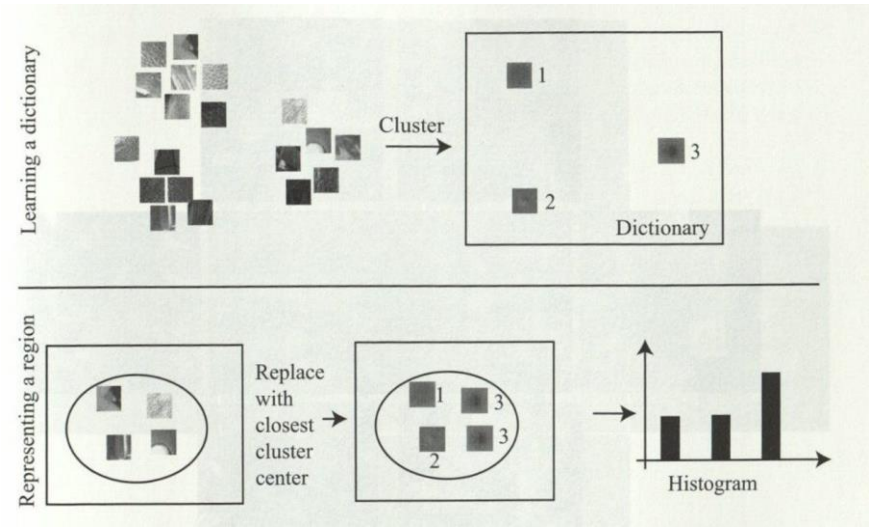
In this whole process of classification of textures, given a mixed texture image, first we will have to apply filters onto the image, here the filters such as the spot filter and the bar filter represent the elements of the texture called textons.

After the obtaining the filter responses from the image, we need to rectify the responses. Rectification is required to enhance the filter outputs, so that the positive and negative responses do not cancel. This can be achieved by halfway rectification.

After this, compute a local summary of the rectified filter output. This can be done by smoothing by using a Gaussian filter. Now we will have the summarized data from the original image (in our case, 16 data channels which will be discussed further in the report).




With this summarized data as the input to the k-means function (Algorithm 6.3), we will achieve the clustered regions having similar texture representation.




The filters can be created by the following functions:

(1) CS5320 spot1, (2) CS5320 spot2, (3) CS5320 bar

**CS5320 spot1**

This spot1 filter is the combination of the 3 Gaussian filters in the following manner:

```
G1 = fspecial('gaussian',11,0.62);
G2 = fspecial('gaussian',11,1);
G3 = fspecial('gaussian',11,1.6);
S1 = G1 - 2*G2 + G3;
```

**CS5320 spot2**

Spot2 filter is a combination of 2 filters. The code followed for this filter is as follows

```
G1 = fspecial('gaussian',11,0.71);
G2 = fspecial('gaussian',11,1.14);
S2 = G1 - G2;
```

**CS5320 bar**

The bar filter uses the oriented Gaussian function. The code for the bar filter is as follows:

```
val = 3;
xmin = -val;
xmax = val;
ymin = -val;
ymax = val;

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);
B = -G1 + 2*G2 - G3;
```

```
B = imresize(B,[11,11]);
```

Two additional filters that are used in this exercise are the mean filter and the variance filter.

The mean filter also called the average filter in Matlab can be accessed in the following manner:

`MN = fspecial('average', hsize)` , here, hsize is the size of the window to be considered for averaging

As the importance of the variance filter is to find the variance at every pixel location, a particular variance filter is not used in the assignment, however the following steps have been followed to obtain the variance of the pixels in the image using the 'stdfilt' function from Matlab which gives the standard deviation of the image. Also to be noted here is that the size of the window for 'stdfilt' function is 11x11.

```
TStDev = stdfilt(im, ones(11));
TVR = TStDev.^2;
```

TVR is the variance of the image im.

At this point we have our filters ready, We now have to find the filter output maps which would have the form $F_i ** im$ for different filters $F_i$.

This filter maps tells us what the window around a pixel looks like. But the texture representation at a point should involve some kind of summary of the nearby filter outputs, rather than just the filter outputs themselves. However, we must process the filter response maps before we summarize. Ex., a light spot filter will give a positive response to a light spot on a darker background. If we simply average filter responses over a patch, then a patch having dark and light spots might record the same near-zero average as a patch containing no spot·. This would be misleading.

Hence, we should report both $\max\left(0, F_i ** im(x,y)\right)$ and $\max\left(0, -F_i ** im(x,y)\right)$ this is called as the half-wave rectification.

After this, we summarize the positive and negative response maps for each filtered data, by smoothening over a neighborhood twice the filter width (twice 11x11 = 22x22).

The process discussed above, just after forming the filters can be implemented into the CS5320_texture_params function whose code is given below.

**CS5320_texture_params**

```
function params = CS5320_texture_params(im)
% CS5320_texture_params - compute texture parameters
% On input:
%     im (mxnx3 array): input image
% On output:
%     params (mxnx16 array): texture parameter image
```

```
%       channel 1: spot1 summarized positive
%       channel 2: spot1 summarized negative
%       channel 3: spot2 summarized positive
%       channel 4: spot2 summarized negative
%       channel 5: bar (0) summarized positive
%       channel 6: bar (0) summarized negative
%       channel 7: bar (45) summarized positive
%       channel 8: bar (45) summarized negative
%       channel 9: bar (90) summarized positive
%       channel 10: bar (90) summarized negative
%       channel 11: bar (135) summarized positive
%       channel 12: bar (135) summarized negative
%       channel 13: mean summarized positive
%       channel 14: mean summarized negative
%       channel 15: variance summarized positive
%       channel 16: variance summarized negative
% Call:
%     p_im = CS5320_texture_params(im_tex);
% Author:
%     Clinton Fernandes
%     UU
%     Spring 2016
%

[rows,cols] = size(im);
params = zeros(rows,cols,16);

%filters
S1 = CS5320_spot1;
S2 = CS5320_spot2;

B90 = CS5320_bar(1,0,0,-1);
B90 = imresize(B90,[101,101]);
B0 = imrotate(B90,90,'crop');
B45 = imrotate(B90,45,'crop');
B135 = imrotate(B45,90,'crop');

B90 = imresize(B90,[11,11]);
B0 = imresize(B0,[11,11]);
B45 = imresize(B45,[11,11]);
B135 = imresize(B135,[11,11]);
MN = fspecial('average', 11);

%response maps
TS1 = filter2(double(S1),im);
TS2 = filter2(S2,im);
TB0 = filter2(B0,im);
TB45 = filter2(B45,im);
TB90 = filter2(B90,im);
TB135 = filter2(B135,im);
TMN = filter2(MN,im);

TStDev = stdfilt(im, ones(11));
TVR = TStDev.^2;
```

```
%rectification
RS1_1 = max(0,TS1);
RS1_2 = max(0,-TS1);
RS2_1 = max(0,TS2);
RS2_2 = max(0,-TS2);
RB0_1 = max(0,TB0);
RB0_2 = max(0,-TB0);
RB45_1 = max(0,TB45);
RB45_2 = max(0,-TB45);
RB90_1 = max(0,TB90);
RB90_2 = max(0,-TB90);
RB135_1 = max(0,TB135);
RB135_2 = max(0,-TB135);
RMN_1 = max(0,TMN);
RMN_2 = max(0,-TMN);
RVR_1 = max(0,TVR);
RVR_2 = max(0,-TVR);

%summarize
G = fspecial('gaussian',22,6);
params(:,:,1) = filter2(G,RS1_1);
params(:,:,2) = filter2(G,RS1_2);
params(:,:,3) = filter2(G,RS2_1);
params(:,:,4) = filter2(G,RS2_2);
params(:,:,5) = filter2(G,RB0_1);
params(:,:,6) = filter2(G,RB0_2);
params(:,:,7) = filter2(G,RB45_1);
params(:,:,8) = filter2(G,RB45_2);
params(:,:,9) = filter2(G,RB90_1);
params(:,:,10) = filter2(G,RB90_2);
params(:,:,11) = filter2(G,RB135_1);
params(:,:,12) = filter2(G,RB135_2);
params(:,:,13) = filter2(G,RMN_1);
params(:,:,14) = filter2(G,RMN_2);
params(:,:,15) = filter2(G,RVR_1);
params(:,:,16) = filter2(G,RVR_2);
```

A texture is a set of textons that repeat in some way. We could find image patches that are common. Also, common vectors of the filter outputs can be found. Vector quantization is a good strategy to do this. Here, we represent a collection of vectors as a histogram of cluster centers. K-means clustering is a good choice to do vector quantization.

For the k-means clustering, the following algorithm can be used:

Choose $k$ data points to act as cluster centers
Until the cluster centers change very little
    Allocate each data point to cluster whose center is nearest.
    Now ensure that every cluster has at least
      one data point; one way to do this is by
      supplying empty clusters with a point chosen at random from
      points far from their cluster center.
    Replace the cluster centers with the mean of the elements
      in their clusters.
end

**Algorithm 6.3:** Clustering by K-Means.

We can use the k-means function of Matlab to do this. The process of clustering by k-means can be shown in the CS5320_k_means_texture function whose code is given as follows:

**CS5320_k_means_texture**

```
[rows,cols,planes] = size(params);
[IDX,C] = kmeans(reshape(params,[rows*cols,planes]),k);
IDX_im = reshape(IDX,[rows,cols]);
clusters = C;
elements = IDX_im;
```

## 3. Verification

**CS5320 spot1**

Output of this function is a 11x11 spot filter

The function can be called and the surf and imagesc command can be used to check whether the output is a spot.

```
>> S1 = CS5320_spot1;
>> surf(S1);
>> imagesc(S1);
```

*Figure: code to call the function*

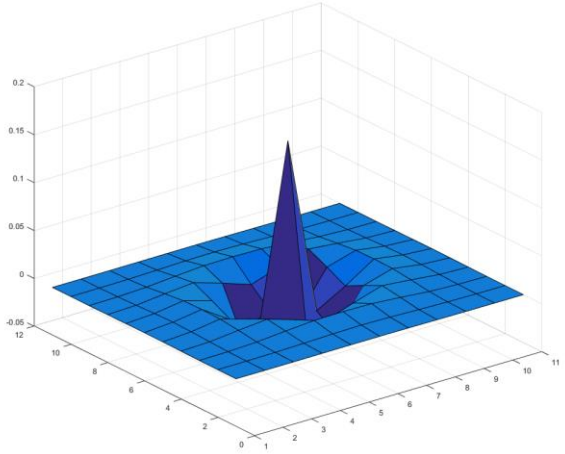


*Figure: Surf(S1)*

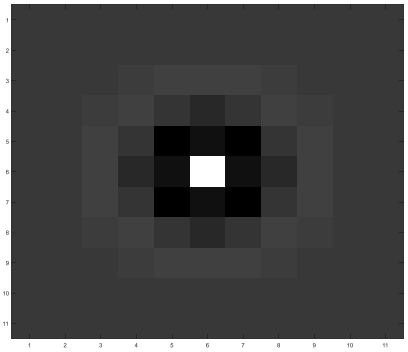


*Figure: imagesc(S1)*

**CS5320 spot2**

Output of this function is a 11x11 spot filter

The function can be called and the surf and imagesc command can be used to check whether the output is a spot.

```
>> S2 = CS5320_spot2;
>> surf(S2);
>> imagesc(S2);
```

*Figure: code to call the function*

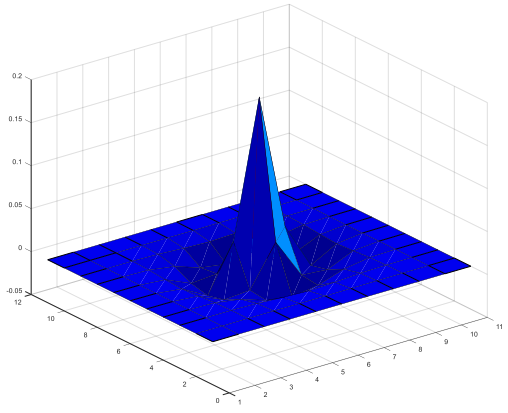


*Figure: Surf(S2)*

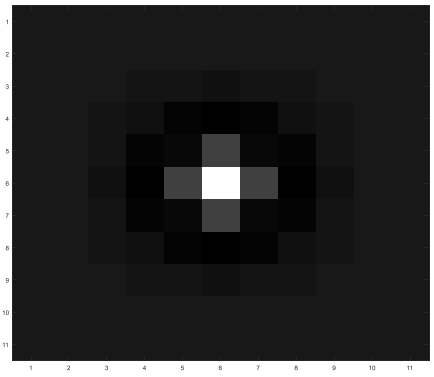


*Figure: imagesc(S2)*

**CS5320_bar**

Output of this function is a 11x11 bar filter

The function can be called and the surf and imagesc command can be used to check whether the output is a bar.

```
>> B = CS5320_bar(1,0,0,-1);
>> surf(B);
>> imagesc(B);
```

*Figure: code to call the function*

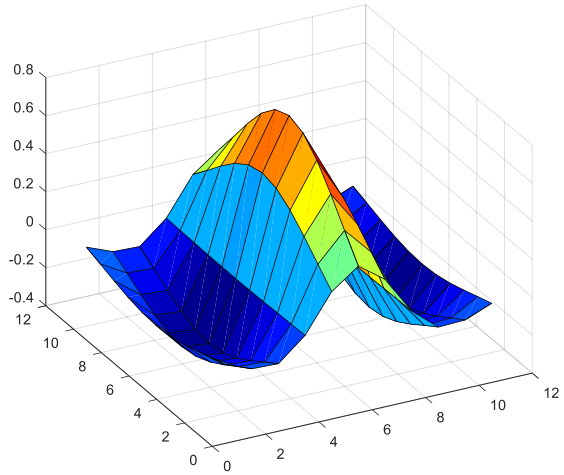



*Figure: Surf(B)*

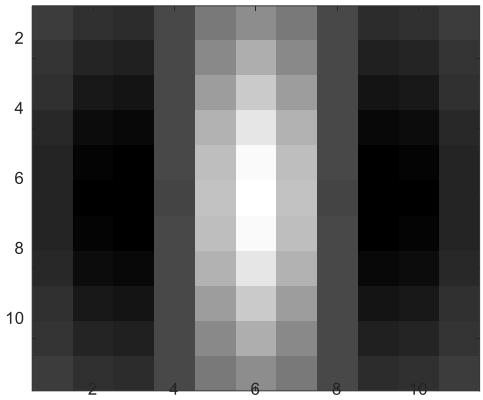



*Figure: imagesc(B)*

**CS5320_texture_params**

The output of the function is p_im (mxnx16 array): texture parameter image.

These 16 data sets are the summarized positive and negative filter responses.

To verify some of the output channels, we can do the following.

For the bar filter at 0 degrees (horizontal), in the current code, the positive summarized response is stored in p_im(:,:,5) and the negative summarized response is stored in p_im(:,:,6).

After applying the CS5320_texture_params function on a test image (figure shown on page 9), we can check whether the output is correct by using the combo function.

```
>> im = rgb2gray(imread('test.jpg'));
>> p_im = CS5320_texture_params(im);
>> N = 5;combo(im,p_im(:,:,N)<max(max(p_im(:,:,N)))*0.8);
>> N = 6;combo(im,p_im(:,:,N)<max(max(p_im(:,:,N)))*0.8);
>> N = 9;combo(im,p_im(:,:,N)<max(max(p_im(:,:,N)))*0.8);
```

N == channel number

Bar filter @ 0 degree data

*Figure: p_im(:,:,5)*

*Figure: p_im(:,:,6)*

Bar filter @ 90 degree data

*Figure: p_im(:,:,9)*

*Figure: p_im(:,:,10)*

Spot1 filter data

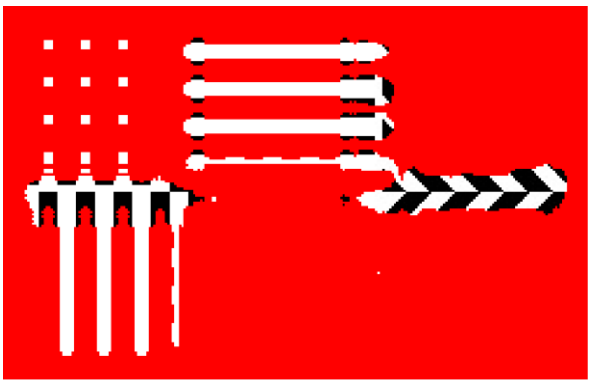


*Figure: p_im(:,:,1)*

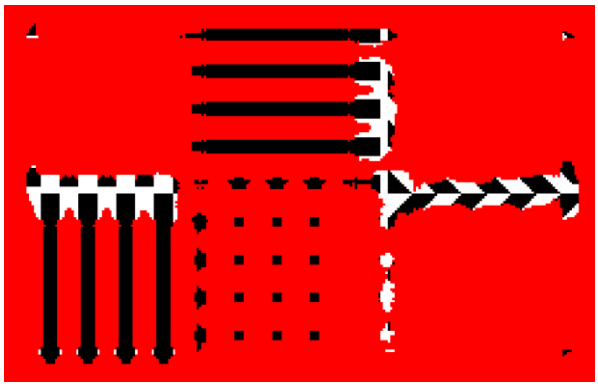


*Figure: p_im(:,:,2)*

To check the mean and the variance filter responses

**CS5320_k_means_texture**

On input:

im (mxnxp array): texture parameter images (p mxn images)

k (int): number of clusters desired

On output, the function gives:

clusters (kxp array): k cluster center vectors

elements (mxn array): cluster index image elements(r,c) is cluster number

To verify whether the function is working properly, we can use the following code:

```
im = rgb2gray(imread('test.jpg'));
p_im = CS5320_texture_params(im);
[cl,el] = CS5320_k_means_texture(p_im,9);
```

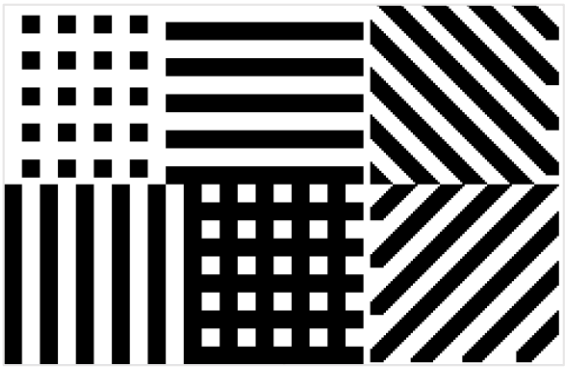


*Figure: 'test.jpg'*

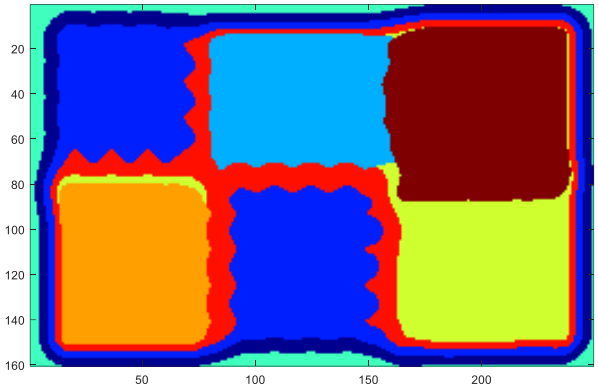



*Figure: imagesc(el)*

*Figure: combo(im,el~=1)*

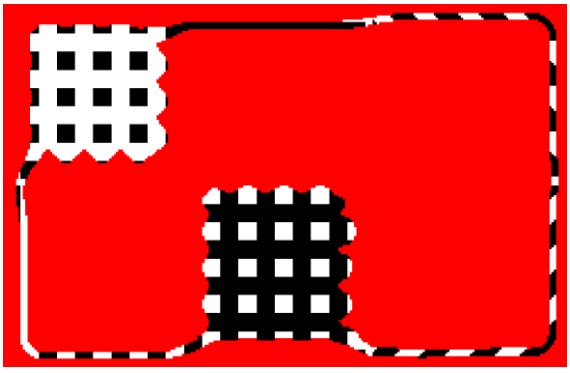


*Figure: combo(im,el~=2)*

*Figure: combo(im,el~=3)*

*Figure: combo(im,el~=5)*

*Figure: combo(im,el~=6)*

*Figure: combo(im,el~=8)*

## 4. Data

1.

Consider that the kmeans function for 10 clusters is applied on the im_tex image, the output; clusters, can be shown in different colors and also the good texture classes can be shown by using combo(im_tex, el~=class);

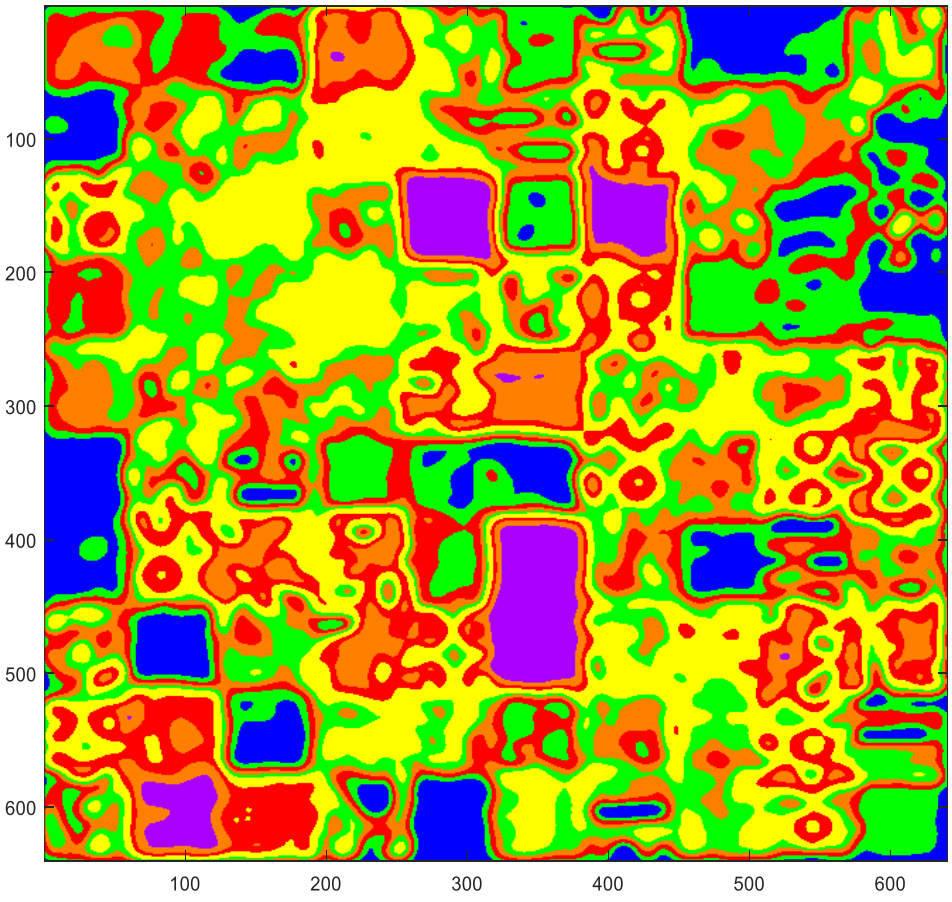
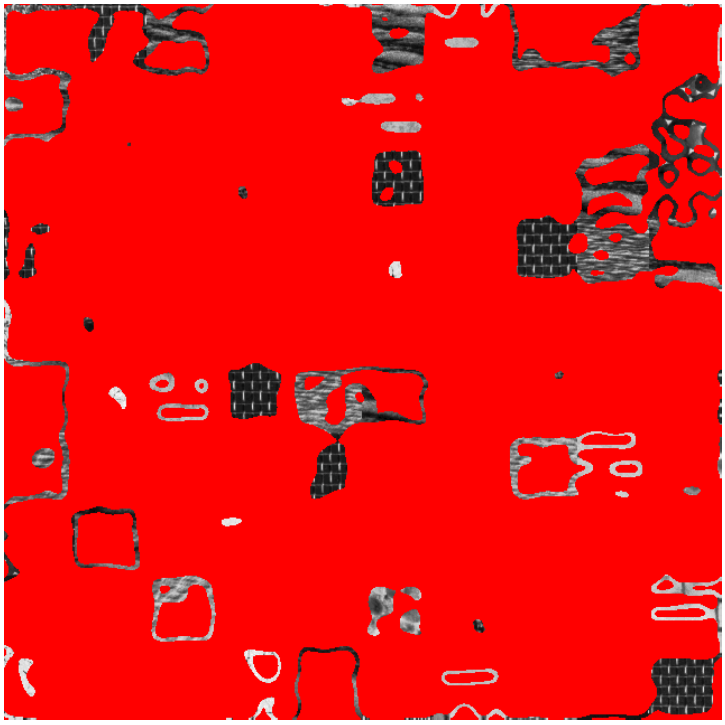
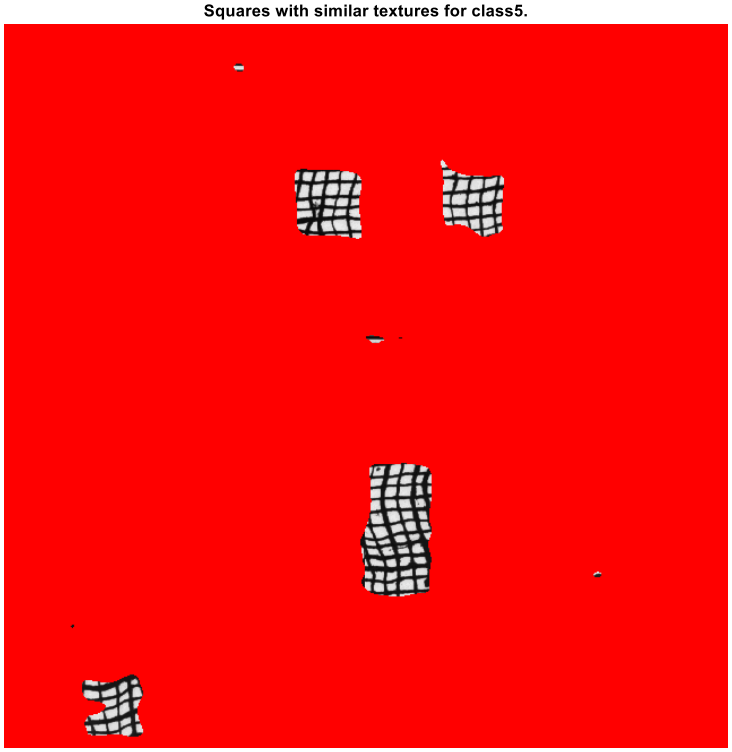



*Figure: pseudo color plot of the clusters*

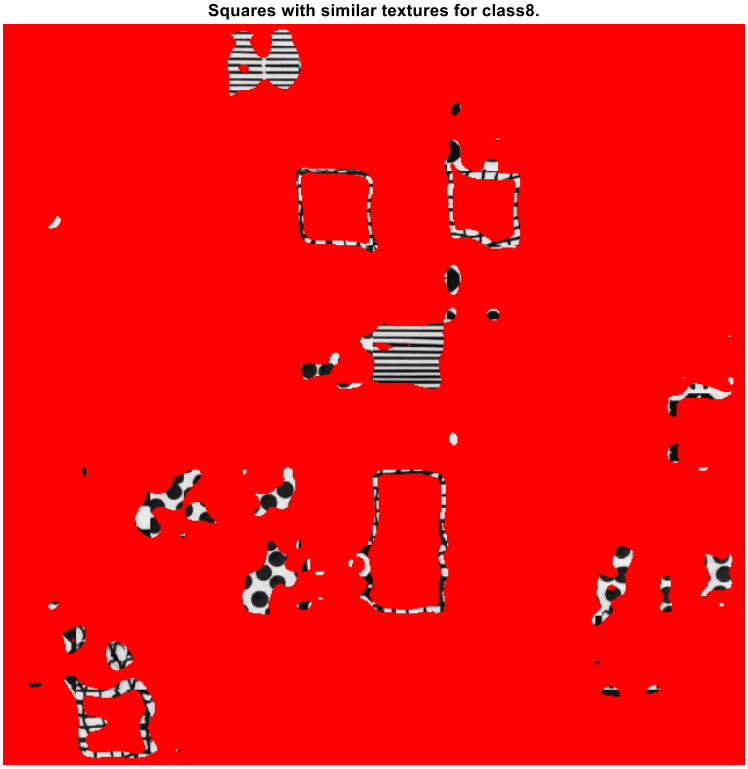
Squares with similar textures for class2.



Squares with similar textures for class5.



Squares with similar textures for class8.



Squares with similar textures for class9.

2.

Applying the kmeans function onto the im_tex mixed texture image, we get the output as

cluster center vectors and cluster index image.

Here, we can vary the number of clusters; input to the kmeans function, and check the output of kmeans clusters by pseudo-colored classification image as well as some of the combo(im_tex,el~=class) results that show good texture squares.

| Number of Clusters considered | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| good texture classes found | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 3 | 3 | 4 | 4 | 6 | 6 | 5 |

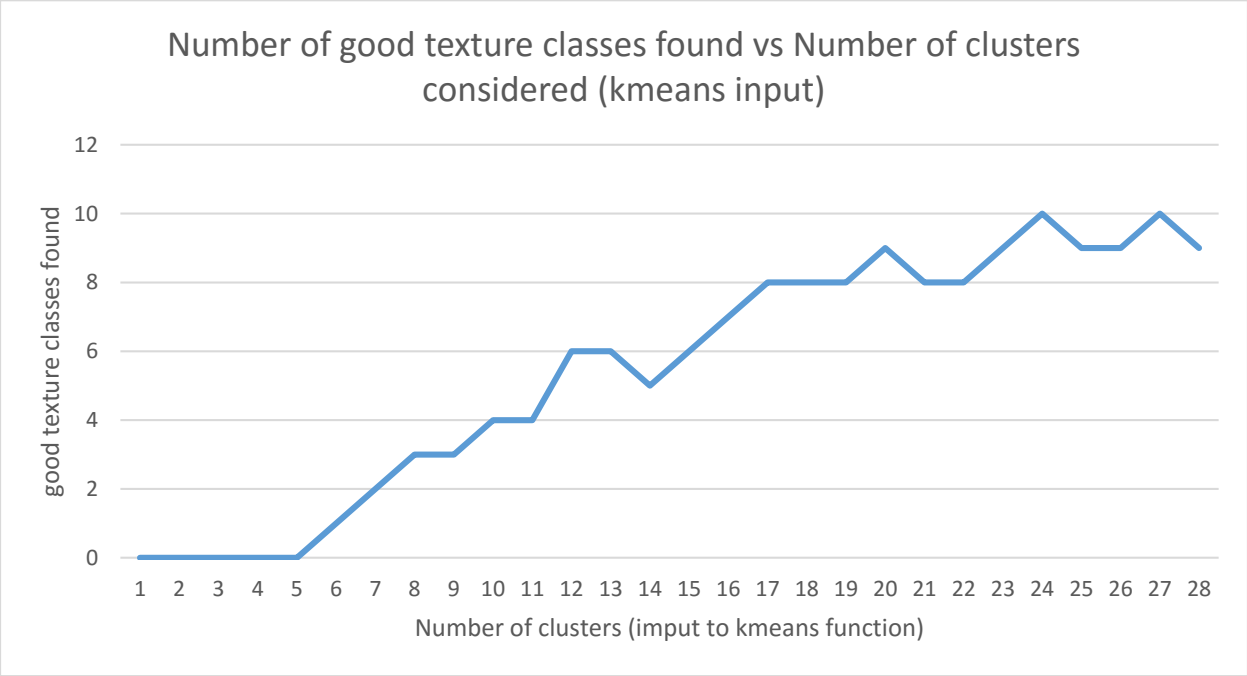
| Number of Clusters considered | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| good texture classes found | 6 | 7 | 8 | 8 | 8 | 9 | 8 | 8 | 9 | 10 | 9 | 9 | 10 | 9 |

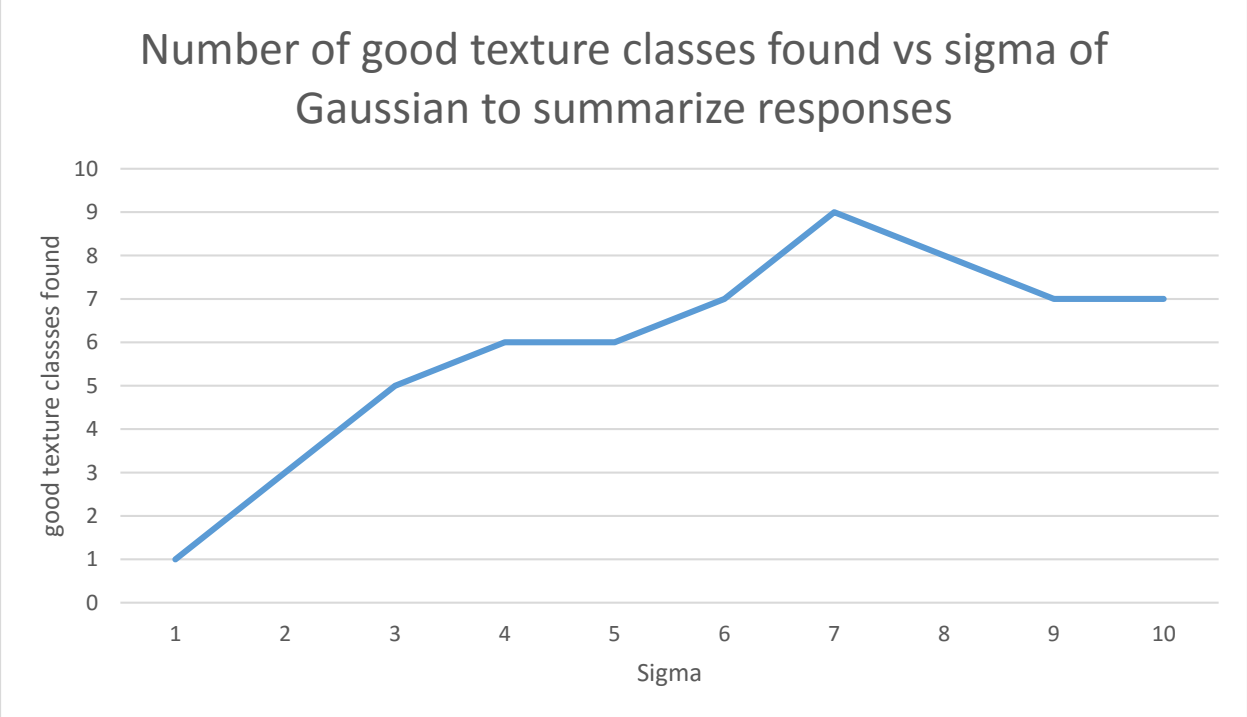
3.

Also, we can keep the number of clusters input of kmeans as constant; 20 and vary the sigma of the Gaussian window used to summarize in the texture_params function.

| sigma of Gaussian to summarize responses | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| good texture classes found | 1 | 3 | 5 | 6 | 6 | 7 | 9 | 8 | 7 | 7 |

## 5. Analysis


Number of good texture classes found vs Number of clusters considered (kmeans input)
good texture classes found
0
2
4
6
8
10
12
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
Number of clusters (imput to kmeans function)


Number of good texture classes found vs sigma of Gaussian to summarize responses
good texture classes found
0
1
2
3
4
5
6
7
8
9
10
1 2 3 4 5 6 7 8 9 10
Sigma

| | mean | variance |
|---|---|---|
| variation of number of clusters | 5.1 | 13 |
| variation of sigma | 5.9 | 5.1 |

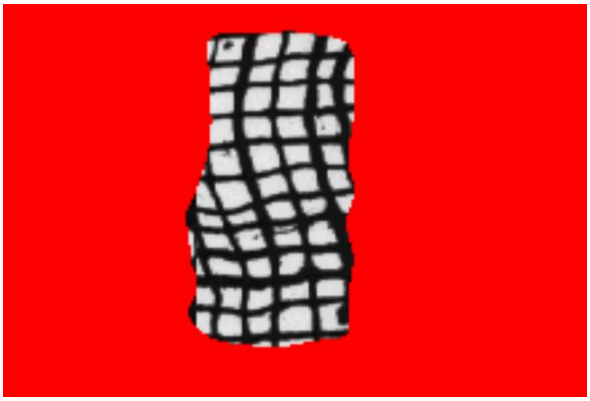
## 6. Interpretation

From the above experiment, it can be interpreted that, as the number of the cluster input to kmeans increases, the number of good texture squares found increases. Good texture is when a favorable number of squares having a particular texture is found. Number of good textures is when different textures are found properly.

Also, the sigma for the Gaussian filter used to summarize the filter responses was varied. It was found that the Number of good textures found increases till about at sigma = 7 and then decreases.

## 7. Critique

I was not able to properly find clusters having whole squares of the textures. Example: if the input to the k-means function was 20 clusters, then it was seen that only about 4-8 cluster squares having similar texture were detected but most of the clusters showed lines in the following manner (figure on the right).



I tried my best to improve the performance of the algorithm by varying different parameters used in the functions, but was able to get just satisfactory results as shown in the figure on the right.

If I had some more time, I would have tried different values of the parameters used in the filter, especially the spot and bar filters. I think that scaling the filter sizes to more than 11x11 would also help in detecting features like big spots, which the normal filters used, were unable to detect.

## 8. Log

Coding/debugging time: 16 hrs

Report: 4.5 hrs