Oblig2 INF4300

mathiaki

16. november 2017

Innhold

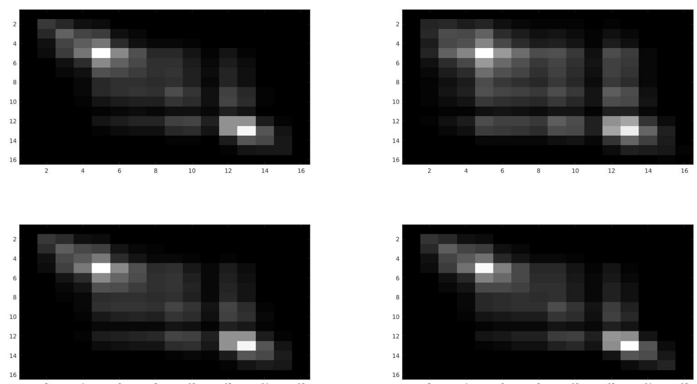
1 Texture description

The first thing to do in this mandatory assignment was to find the best GLCM for all 3 of the textures. In the last assignment our job was to find these matrices, but this time we already have the finished glcm matrices.

With the finished GLCM matices, we can now get the feature images for the different orientations.

1.1 Matrix data

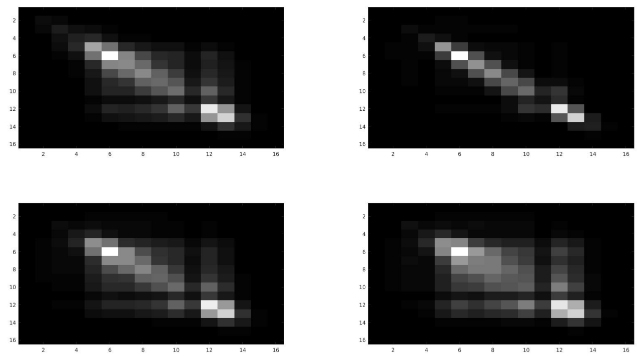
from the files included we get this result for the 4 different textures:



(a) Texture 1

Figur 1: Texture 1 GLCM

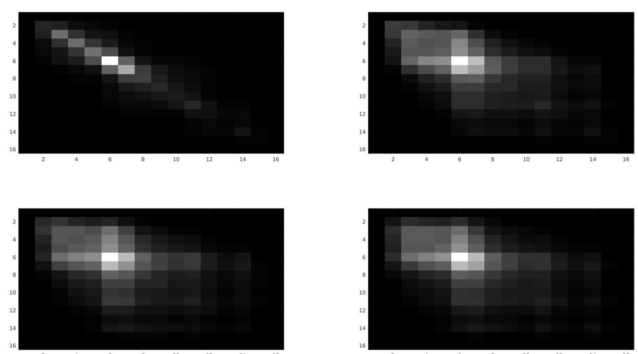
From this first texture we choose the second of the 4 images.
This corresponds to dx=1 dy=0



(a) Texture 2

Figur 2: Texture 2 GLCM

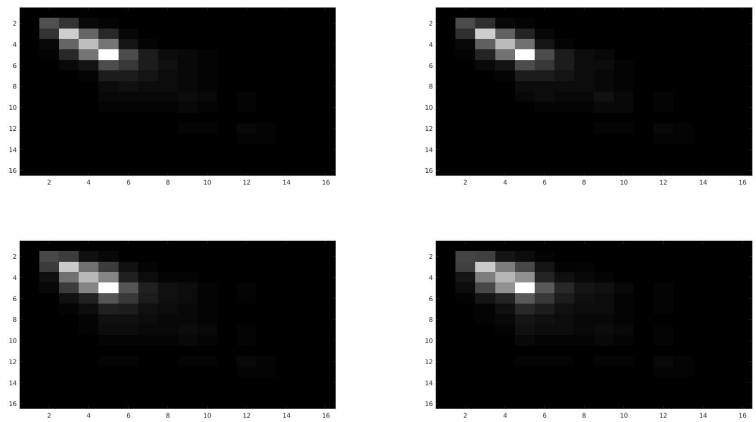
From this first texture we choose the first of the 4 images.
This corresponds to dx=0 dy=-1



(a) Texture 3

Figur 3: Texture 3 GLCM

From this first texture we choose the first or second of the 4 images.
This corresponds *best* to dx=0 dy=-1



(a) Texture 4

Figur 4: Texture 4 GLCM

From this first texture we choose the first of the 4 images.
This corresponds to dx=0 dy=-1
Now that we have all the necessary dxy values, we can now start with the sliding GLCM part of the program.

From this point onwards we stop using the GLCM matrices from the assignment, and start making and using our own:

```
function glcm = GLCM(img, G, dx, dy)
% at this pont in the process this function was not 100% self-made.
% Inspiration for Kristoffer Hoiseter, since he did the first
% obligatory assignment in MATLAB, and I made my gliding window in
    python.


%size of image
[M,N] = size(img);


W = 1./((M-dx)*(N-dy));
glcm = zeros(G);

%going through and counting
for i=1:M
    for j=1:N
        %making sure the indexes does not exceed matrix dimensions
        if j + dy < 1 || j + dy > N || i + dx < 1 || i + dx > M
            continue;
        else
            a = img(i,j);
            b = img(i+dx,j+dy);
            glcm(a+1,b+1) = glcm(a+1,b+1) + 1;
        end
    end
end

%symmetric and normalized
glcm = glcm + glcm';
glcm = glcm./sum(sum(glcm));
%glcm = W*glcm;
```

An important note here is that the GLCM is going to be symmetric, so Q2 and Q3 will be (close to) identical every time.

2 Quadrant and sliding

With the GLCM matrices from assignment 1 we can now divide the each of the GLCM matrices in to 4 parts:

```
GLCM Gliding window
|---------------|
|Q11|Q12|       |
|---|---|  Q2   |
|Q13|Q14|       |
|-------|-------|
|       |       |
|  Q3   |  Q4   |
|       |       |
|---------------|
```

Figur 5: GLCM Gliding matrix

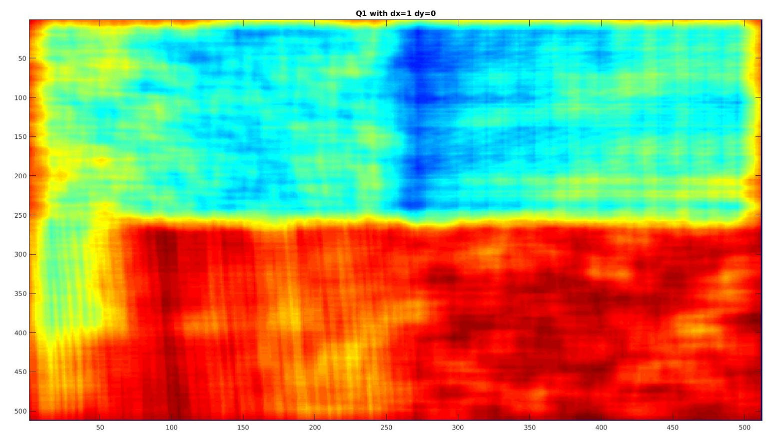
When we now run the gliding GLCM the result of the different quadrants are stored in the respective variables. As shown in 5.

As the Q1 quadrant has the most difference between the different textures, it is natural to spit the quadrant up in to 4 subquadrants.

```
function [Q1,Q2,Q3,Q4,Q11,Q12,Q13,Q14] = gGLCM(img, G, dx, dy, window)
% at this pont in the process this function was not 100% self-made.
% Inspiration for Kristoffer Hoiseter, since he did the first
% obligatory assignment in MATLAB, and I made my gliding window in python.


[M,N] = size(img);
halfWindow = floor(window/2);

%expanding the image with a border
imgBorder = zeros(M+window-1, N+window-1);
imgBorder(halfWindow+1:end-halfWindow,halfWindow+1:end-halfWindow) = img;

%size of the new image
[Mborder, Nborder] = size(imgBorder);

i = repmat((0:(G-1))', 1, G);
j = repmat((0:(G-1)), G, 1);


Q1 = zeros(M,N);
Q2 = zeros(M,N);
Q3 = zeros(M,N);
Q4 = zeros(M,N);

%spitting the top left quadrant in 4, because the action is happening here
```

```
Q11 = zeros(M,N);
Q12 = zeros(M,N);
Q13 = zeros(M,N);
Q14 = zeros(M,N);


%going through the image
for m = 1+halfWindow:Mborder-halfWindow-1;
     for n = 1+halfWindow:Nborder-halfWindow-1;

          win = imgBorder(m-halfWindow:m+halfWindow, n-halfWindow:n+halfWindow);

          p = GLCM(win,G,dx,dy);

          Q1(m-halfWindow,n-halfWindow)=sum(sum(p(1:G/2,1:G/2)))/sum(sum(p));
          Q2(m-halfWindow,n-halfWindow)=sum(sum(p(1:G/2,G/2:G)))/sum(sum(p));
          Q3(m-halfWindow,n-halfWindow)=sum(sum(p(G/2:G,1:G/2)))/sum(sum(p));
          Q4(m-halfWindow,n-halfWindow)=sum(sum(p(G/2:G,G/2:G)))/sum(sum(p));

          Q11(m-halfWindow,n-halfWindow)=sum(sum(p(1:G/4,1:G/4)))/sum(sum(p));
          Q12(m-halfWindow,n-halfWindow)=sum(sum(p(1:G/4,1+G/4:G/2)))/sum(sum(p));
          Q13(m-halfWindow,n-halfWindow)=sum(sum(p(1+G/4:G/2,1:G/4)))/sum(sum(p));
          Q14(m-halfWindow,n-halfWindow)=sum(sum(p(1+G/4:G/2,1+G/4:G/2)))/sum(sum(p));


     end


end
end
```

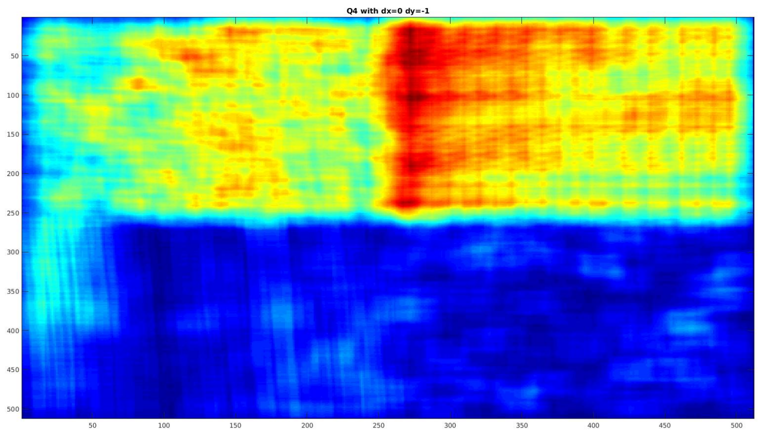
The Q matrices can now be analyzed:



(a) Texture 4

Figur 6: Texture 4 GLCM

Instead of showing all 8 sliding window GLCM matrices, I have chosen to only show the 3 that i used throughout the rest of the assignment.



(a) Texture 4

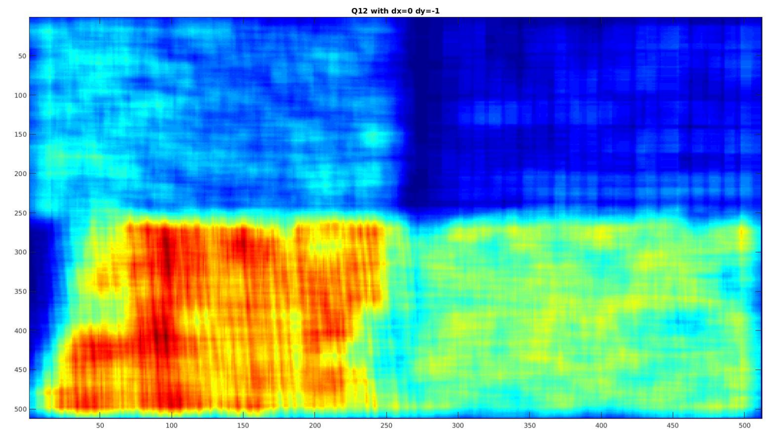
Figur 7: Texture 4 GLCM

3 Multivariate Gaussian Classifier

After we got the GLCM matrix for the 3 different values, our next next task is to use a Multivariate Gaussian Classifier to try to classify the 4 patterns.

The formula for Multivariate Gaussian Classifier (in multiple dimensions) is:

$$y = f(x, \mu, \sum) = \frac{1}{\sqrt{|\sum|(2\pi)^d}} exp\left(-\frac{1}{2}(x-\mu)\sum^{-1}(x-\mu)'\right)$$



(a) Texture 4

Figur 8: Texture 4 GLCM

4 Classification

this is classification

5 Testing

this is testing

5.1 The discarded result

Getting a good result varies a lot on the GLCM matrix that we chose early in the process. This is a result that i chose to discard, where the GLCM matrices had different values than the one i ended up with.

```
confusion =
       56294        1661        4562         513
         378       58446         725          36
        8353        5428       53798       15786
           0           0        6450       49714
acc =
   83.2565

confusion =
       56610        2273        2883         522
        1205       54745         267           0
        7210        8517       56017       14861
           0           0        6368       50666
acc =
   83.1749

confusion =
       39056       18547       15736        9122
       12458       38550        3086       25138
       13511        8438       46573       31637
           0           0         140         152
acc =
   47.4285
```

Figur 9: Result from a run with dx=0 dy=-1 and dx=1 dy=0

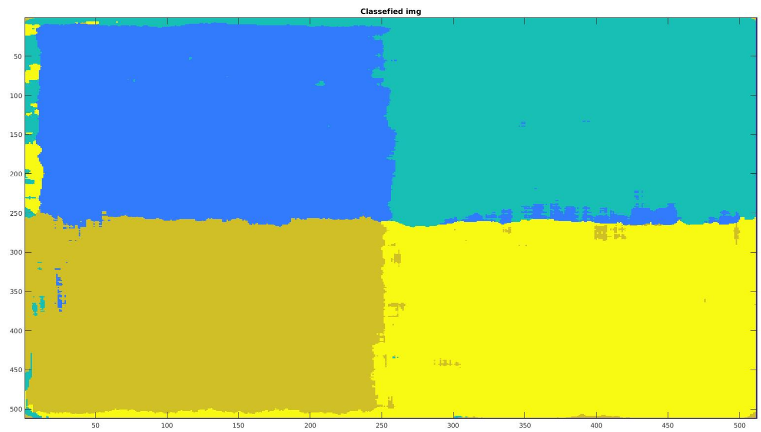
Here we have almost 50% on the last test, which is higher than the one that i ended up with, but it failed to find any of the pattern from the bottom right corner.

5.2 The final result

The final result yielded this result:

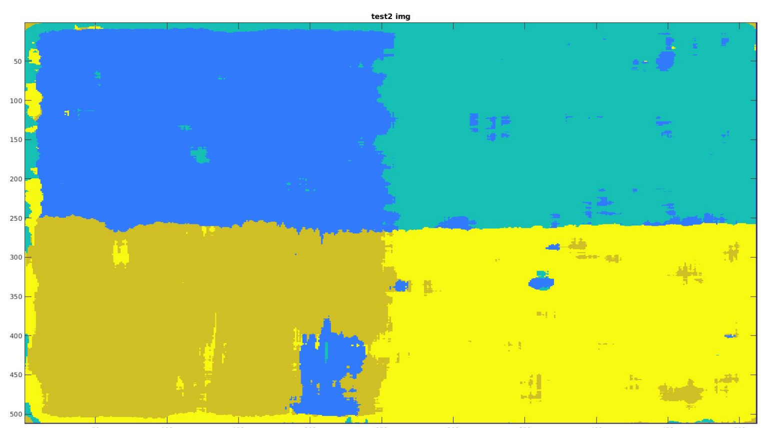
```
confusion =
           0           0         255         255         513
           0       59360        1594        1034        1070
           0        4223       63678         317         724
           0         140           8       60413         788
           0        1302           0        3516       62954
acc =
   93.9960

confusion =
           0           0         255         255         513
           0       58573        1866        5321                  857
           0        4824       63374         401            1024
           0         415          24       54001            2084
           0        1213          16        5557              61571
acc =
   90.6063

confusion =
           0           0         255         255         513
           0       55720       44913       15930        4903
           0        8506       19448        5213       46792
           0         357          32       38357         305
           0         442         887        5780       13536
acc =
   48.4699
```

Figur 10: Result from a run with dx=1 dy=0 and dx=0 dy=-1 (opposite of the other result)



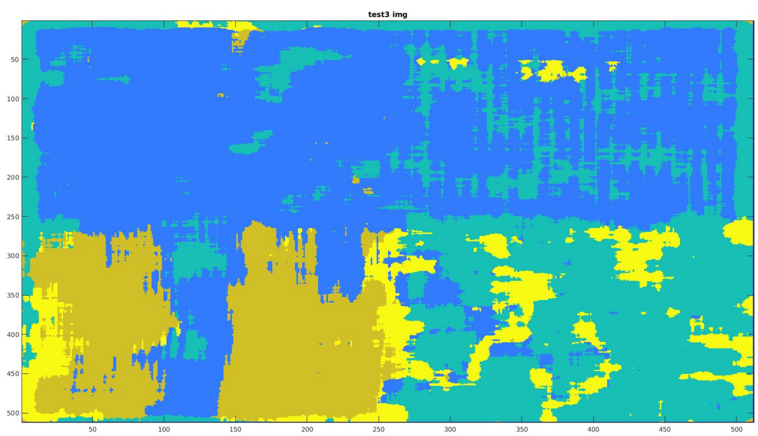
(a) Texture 4

Figur 11: Final classification after training



(a) Texture 4

Figur 12: Test classification on test number 1



(a) Texture 4

Figur 13: Test classification on test number 2