Oblig2 INF4300

mathiaki

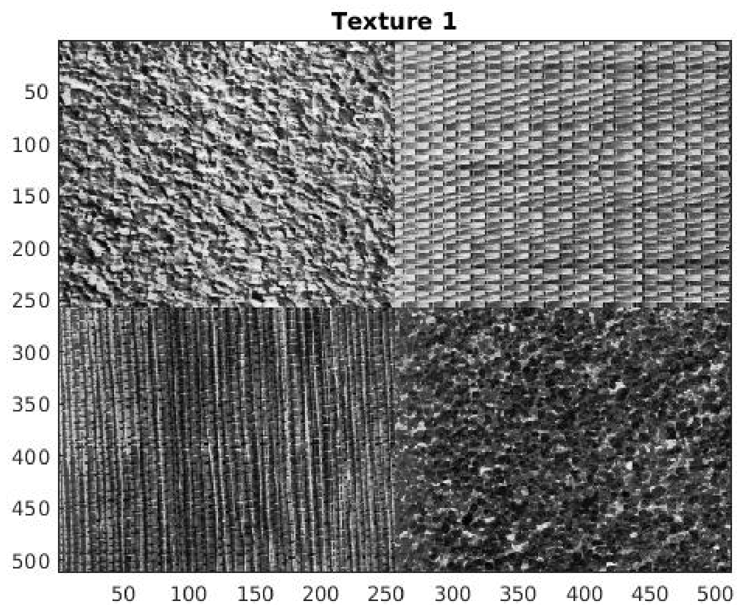
17. november 2017

Innhold

1 Texture description

In this assignment our first task is to find the best gray level co occurrence matrix (GLCM) for the four textures given in the assignment description. In the previous assignment our task 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.

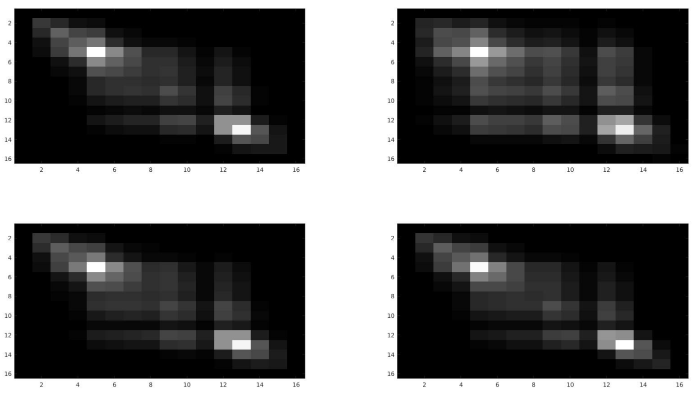


(a) Texture

Figur 1: Original texture

1.1 Matrix data

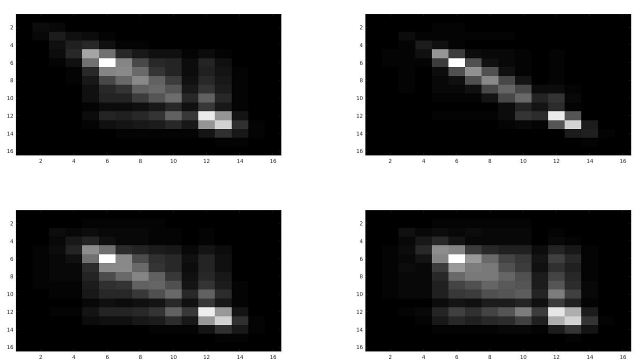
Included in the assignment we got some matrices with the GLCM for each texture. We want to find the GLCM matrices with the best pattern. Like strong points, or clear lines.



(a) Texture 1

Figur 2: Texture 1 GLCM

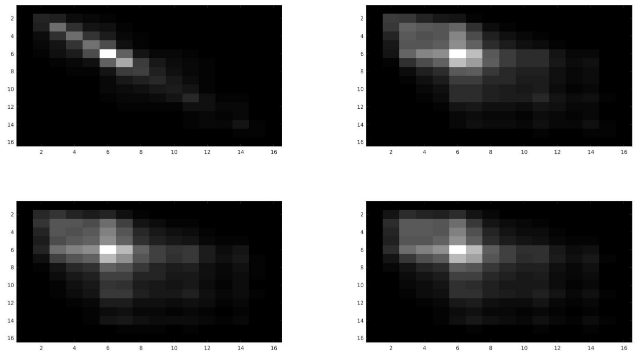
From this first texture we choose the first of the 4 images.
This corresponds to dx=1 dy=0



(a) Texture 2

Figur 3: Texture 2 GLCM

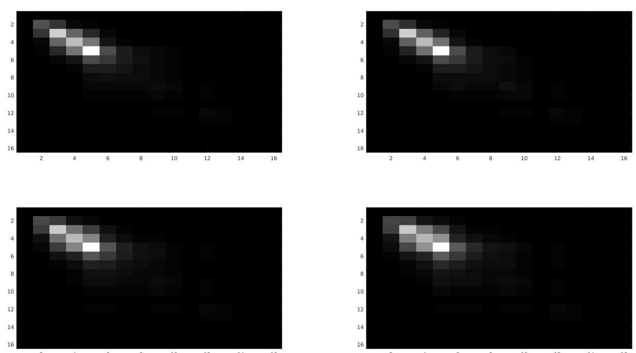
From this first texture we choose the second of the 4 images.
This corresponds to dx=0 dy=-1



(a) Texture 3

Figur 4: Texture 3 GLCM

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



(a) Texture 4

Figur 5: Texture 4 GLCM

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

With all the necessary dxy values, we can 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)
%This function was not 100% self-made.
% Inspiration for Kristoffer Hoiseter, and the weekly assignments, since 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. The same is true for Q12 and Q13.

2 Quadrant and sliding

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

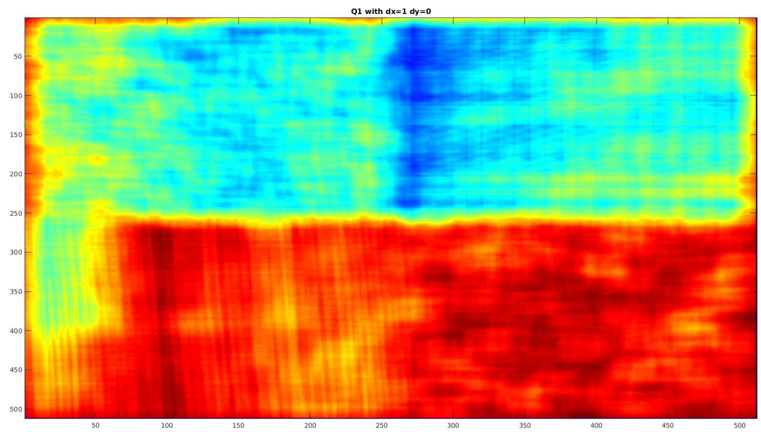
```
GLCM Gliding window
|---------------|
|Q11|Q12|       |
|---|---|  Q2   |
|Q13|Q14|       |
|-------|-------|
|       |       |
|  Q3   |  Q4   |
|       |       |
|---------------|
```

Figur 6: GLCM Gliding matrix

When we run the gliding GLCM the result of the different quadrants are stored in the respective variables. As shown in 6.

Since the Q1 quadrant has the most varience of the different textures, it is natural to split the quadrant in to 4 subquadrants. The gliding GLCM can be found in appendix A **??**.

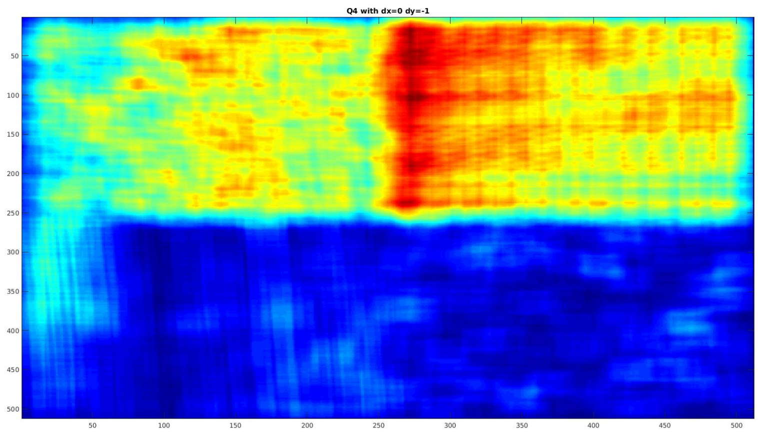
The Q matrices was then analyzed. And at this point we had to choose how many features we want to use in the classification. We ended up with 3 features, and subsequently 3 quadrants were used:
Instead of showing all 8 sliding window GLCM matrices, I have chosen to only show the 3 that i used throughout the rest of the assignment.



(a) Quadrant Q1

Figur 7: Qadrant Q1 with dx=1 and dy=0

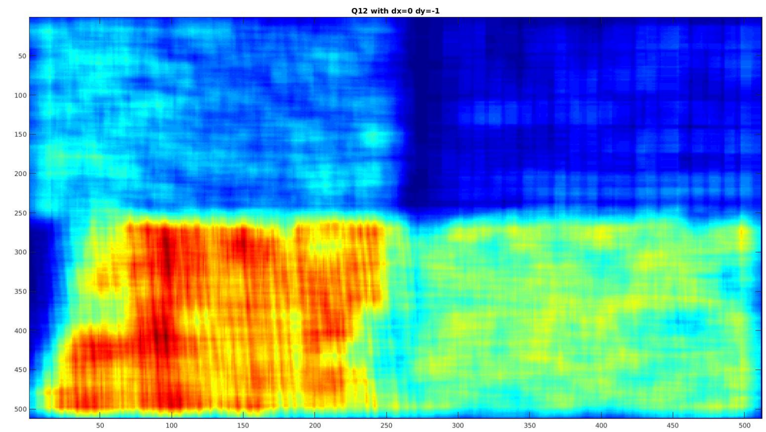
The first quadrant is the top left in the GLCM. As expected is the 2 textures with the most *black* to *black* pixels are *hot*.



(a) Quadrant Q4

Figur 8: Qadrant Q4 with dx=0 and dy=-1

The second quadrant is almost the opposite, of the first. Here the GLCM jumps from high to high values, as it does in Q4. We have here a horizontal pattern.



(a) Quadrant Q12

Figur 9: TQadrant Q12 with dx=0 and dy=-1

The last quadrant is chosen mainly for the recognition of the bottom left texture.
We now have 3 features to differentiate the textures. We can work on the

Multivariate Gaussian Classifier to train and recognize the textures.

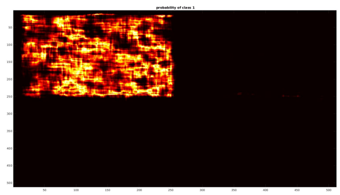
3 Multivariate Gaussian Classifier

After we got the GLCM matrix for the 3 different values, our next task is to use a Multivariate Gaussian Classifier to try to classify the 4 patterns.
After the features was identified we ran it on the training mask to extract the μ and σ.
μ is the mean of the classes, and σ use the covariance of all the features for all 4 classes.
With these values we can calculate the Multivariate Gaussian Classifier. The formula for Multivariate Gaussian Classifier (in multiple dimensions) is:

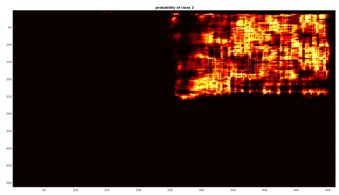
$$y = f(x, \mu, \sigma) = \frac{1}{\sqrt{|\sigma|(2\pi)^d}} exp\left(-\frac{1}{2}(x-\mu)\sum^{-1}(x-\mu)'\right)$$

4 Classification

After running the classifier on the first training set the result was this:

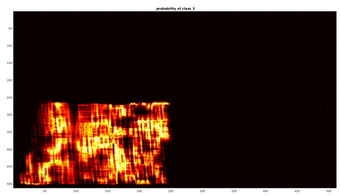


(a) Probability of class 1

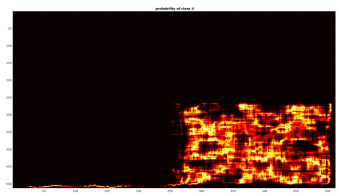


(b) Probability of class 2

Figur 10: 2 first classifiers



(a) Probability of class 3



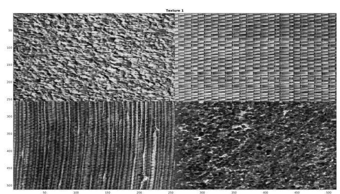
(b) Probability of class 4

Figur 11: 2 last classifiers

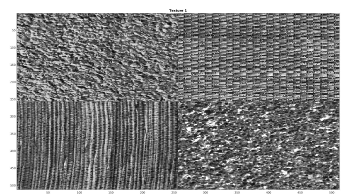
5 Testing

5.1 The final result

The last part is to test the program on the 2 testing sets:



(a) Test figure 2



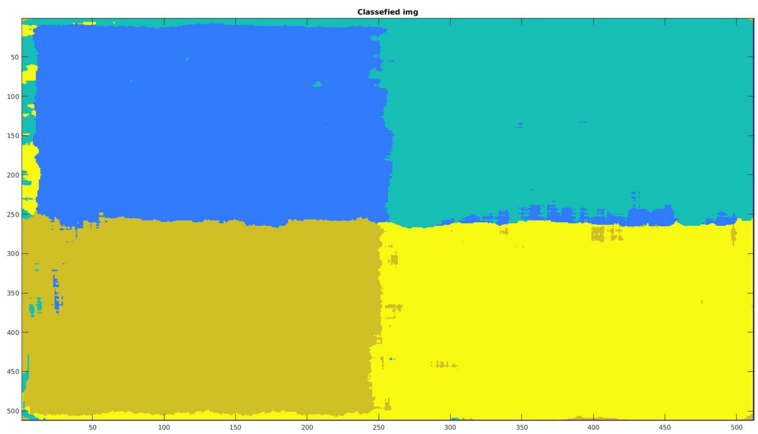
(b) Test figure 3

Figur 12: The two test figures

We would expect the classifier to find the 4 classes pretty well on both the images, with the best result on figure 2, since it hasn't as much artifacts compared to test figure 3.
The dx/dy values with the chosen quadrants yielded the following result:

5.2 Training set

The training set gave us



(a) Texture 4

Figur 13: Final classification after training

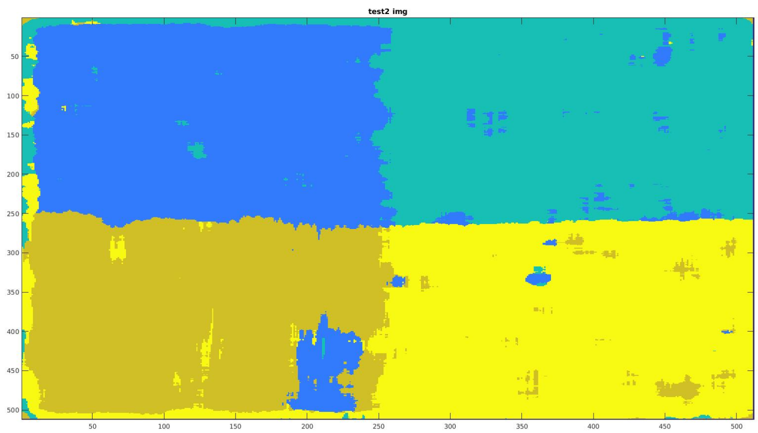
Confusion Matrix

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

Tabell 1: Accuracy: 93.9960%

As we can see, the training data is close to perfect, as expected.

5.3 Test set 1



(a) Texture 4

Figur 14: Test classification on test number 1

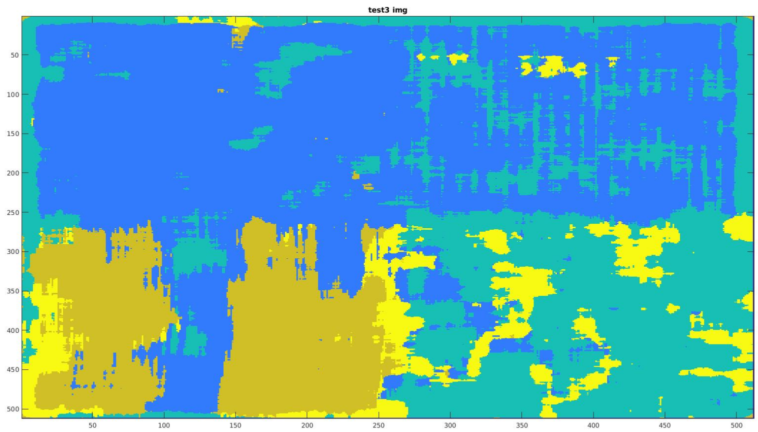
Confusion Matrix

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

Tabell 2: Accuracy: 90.6063%

Here we got some trouble in class 3. Everything else is really close. Good enough result.

5.4 Test set 2



(a) Texture 4

Figur 15: Test classification on test number 2

Confusion Matrix

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

Tabell 3: Accuracy: 48.4699%

At this point the test is to warped to get a desirable result. We end up at almost 50 percent, which isn't too bad.

6 appendix A - Gliding GLCM

First, the gliding GLCM

```
function [Q1,Q2,Q3,Q4,Q11,Q12,Q13,Q14] = gGLCM(img, G, dx, dy, window)
% This function was not 100% self-made.
% Inspiration for Kristoffer Hoiseter, and the weekly assignments, since 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
```

7 appendix B - Main program

```
clear all
close all

%% Oppgave 1
G=16;
t=openmat('mosaic1_train.mat');
t=t.mosaic1_train;
t2=openmat('mosaic2_test.mat');
t2=t2.mosaic2_test;
t3=openmat('mosaic3_test.mat');
t3=t3.mosaic3_test;

tm=openmat('training_mask.mat');
tm=tm.training_mask;

%making the images easyer to work with
t = uint8(round(double(t) * (G-1) / double(max(t(:)))));
t2 = uint8(round(double(t2) * (G-1) / double(max(t2(:)))));
t3 = uint8(round(double(t3) * (G-1) / double(max(t3(:)))));

figure(1);clf
imagesc(t)
colormap gray
title('Texture 1');


%% print tm

figure(1);clf
imagesc(t)
colormap gray
title('Texture 1');


%% oppgave 1-2
s1=openmat('texture1dx0dymin1.mat');
s1=s1.texture1dx0dymin1;
s2=openmat('texture1dx1dymin1.mat');
s2=s2.texture1dx1dymin1;
s3=openmat('texture1dxmin1dymin1.mat');
s3=s3.texture1dxmin1dymin1;
s4=openmat('texture1dxplus1dy0.mat');
s4=s4.texture1dx1dy0;
s5=openmat('texture2dx0dymin1.mat');
s5=s5.texture2dx0dymin1;
s6=openmat('texture2dxplus1dy0.mat');
s6=s6.texture2dx1dy0;
s7=openmat('texture2dxplus1dymin1.mat');
s7=s7.texture2dx1dymin1;
s8=openmat('texture2dxmin1dymin1.mat');
s8=s8.texture2dxmin1dymin1;
s9=openmat('texture3dx0dymin1.mat');
s9=s9.texture3dx0dymin1;
s10=openmat('texture3dxplus1dy0.mat');
s10=s10.texture3dx1dy0;
```

```
s11=openmat('texture3dxplus1dymin1.mat');
s11=s11.texture3dx1dymin1;
s12=openmat('texture3dxmin1dymin1.mat');
s12=s12.texture3dxmin1dymin1;
s13=openmat('texture4dx0dymin1.mat');
s13=s13.texture4dx0dymin1;
s14=openmat('texture4dxplus1dy0.mat');
s14=s14.texture4dx1dy0;
s15=openmat('texture4dxplus1dymin1.mat');
s15=s15.texture4dx1dymin1;
s16=openmat('texture4dxmin1dymin1.mat');
s16=s16.texture4dxmin1dymin1;


figure(2);clf
title('Texture 1');
colormap gray
subplot(2,2,1)
imagesc(s1)
subplot(2,2,2)
imagesc(s2)
subplot(2,2,3)
imagesc(s3)
subplot(2,2,4)
imagesc(s4)

figure(3);clf
title('Texture 1');
colormap gray
subplot(2,2,1)
imagesc(s5)
subplot(2,2,2)
imagesc(s6)
subplot(2,2,3)
imagesc(s7)
subplot(2,2,4)
imagesc(s8)

figure(4);clf
title('Texture 1');
colormap gray
subplot(2,2,1)
imagesc(s9)
subplot(2,2,2)
imagesc(s10)
subplot(2,2,3)
imagesc(s11)
subplot(2,2,4)
imagesc(s12)

figure(5);clf
title('Texture 1');
colormap gray
subplot(2,2,1)
imagesc(s13)
subplot(2,2,2)
imagesc(s14)
```

```
subplot(2,2,3)
imagesc(s15)
subplot(2,2,4)
imagesc(s16)



%% Oppgave 2

% running gliding GLCM with dx=1 and dy=0 and keeping Q1
[tmpQ1,tmpQ2,tmpQ3,tmpQ4,tmpQ11,tmpQ12,tmpQ13,tmpQ14] = gGLCM(t,G
    ,1,0,31);
Q1=tmpQ1;
% running gliding GLCM with dx=0 and dy=-1 and keeping Q12 and Q4
[tmpQ1,tmpQ2,tmpQ3,tmpQ4,tmpQ11,tmpQ12,tmpQ13,tmpQ14] = gGLCM(t,G
    ,0,-1,31);
Q4=tmpQ4;
Q12=tmpQ12;


%test sets: keeping the same values from the 2 training sets
[TMPt2Q1,TMPt2Q2,TMPt2Q3,TMPt2Q4,TMPt2Q11,TMPt2Q12,TMPt2Q13,
    TMPt2Q14] = gGLCM(t2,G,1,0,31);
t2Q1=TMPt2Q1;
[TMPt2Q1,TMPt2Q2,TMPt2Q3,TMPt2Q4,TMPt2Q11,TMPt2Q12,TMPt2Q13,
    TMPt2Q14] = gGLCM(t2,G,0,-1,31);
t2Q4=TMPt2Q4;
t2Q12=TMPt2Q12;

[TMPt3Q1,TMPt3Q2,TMPt3Q3,TMPt3Q4,TMPt3Q11,TMPt3Q12,TMPt3Q13,
    TMPt3Q14] = gGLCM(t3,G,1,0,31);
t3Q1=TMPt3Q1;
[TMPt3Q1,TMPt3Q2,TMPt3Q3,TMPt3Q4,TMPt3Q11,TMPt3Q12,TMPt3Q13,
    TMPt3Q14] = gGLCM(t3,G,0,-1,31);
t3Q4=TMPt3Q4;
t3Q12=TMPt3Q12;


%% oppgave 3
% Print of the 3 Quadrants of the GLCM i used
figure(6);clf
imagesc(Q1)
colormap jet
title('Q1 with dx=1 dy=0');

figure(7);clf
imagesc(Q4)
colormap jet
title('Q4 with dx=0 dy=-1');

figure(8);clf
imagesc(Q12)
colormap jet
title('Q12 with dx=0 dy=-1');
```

```
%% Oppgave 4
%we got 4 classes and 3 features we use
cl=4;
feature=3;
% actual classes needs to be made.
%Here we make a new ACtualCLass as a mask.
accl = zeros(512,512);
accl(1:512/2, 1:512/2) = 1;
accl(1:512/2, 512/2:512) = 2;
accl(512/2:512, 1:512/2) = 3;
accl(512/2:512, 512/2:512) = 4;




%calculating mu and sigma for each feature*class
my = zeros(cl,feature);
sigma = zeros(feature,feature,cl);


%flatten the training mask
[tmp1 , tmp2] = size(tm);
mask = reshape(tm, tmp1 * tmp2, 1);

%get the 3 feture images
mysigma(1,:)=Q1(:);
mysigma(2,:)=Q4(:);
mysigma(3,:)=Q12(:);
%filling sigma and my based on Q
for c = 1:cl
    for i = 1:feature
        tmp(:, 1) = mysigma(i, :);
        my(c,i) = mean(rot90(tmp(mask == c)));
    end
    sigma(:, :, c) = cov(rot90(mysigma(:,mask == c)));
end



%%my and sigma is used in all 3 runs under.
%% Oppgave 5
f_all(1,:,:) = Q1;
f_all(2,:,:) = Q4;
f_all(3,:,:) = Q12;
[acc,outimg,confusion] = oppg7(f_all, my, sigma, cl, accl);


figure(15);clf
imagesc(outimg)
title('Classefied img');

confusion
acc

%% oppg6.1
```

```


f_all(1,:,:) = t2Q1;
f_all(2,:,:) = t2Q4;
f_all(3,:,:) = t2Q12;
[acc,outimg,confusion] = oppg7(f_all, my, sigma, cl, accl);


figure(16);clf
imagesc(outimg)
title('test2 img');

confusion
acc


%% oppg6.2
f_all(1,:,:) = t3Q1;
f_all(2,:,:) = t3Q4;
f_all(3,:,:) = t3Q12;
[acc,outimg,confusion] = oppg7(f_all, my, sigma, cl, accl);


figure(76);clf
imagesc(outimg)
title('test3 img');

confusion
acc
```

8 appendix C - Classifier

Third, the classifier

```
function [ acc,outimg,confusion ] = oppg7( f_all, my, sigma, cl, actual_matrix)

%first we make the array that will hold the values for all the calculated
%values fir the Multivariate Normal Distribution
MultivariateNormalDistribution = zeros(cl, 512, 512);
for c = 1:cl %for each class
    d = 1/sqrt((2 * pi)^3 * det(sigma(:,:,c))); %denominator
    for m = 1:512 %for each x pixel
        for n = 1:512 %for each y pixel
            MultivariateNormalDistribution(c, m, n) = d*exp(-0.5 * (rot90(f_all(:,m,n)) - my(c,:)) / sigma(:,:,c) * transpose(rot90(f_all(:,m,n)) - my(c,:)));
        end
    end
end

chance = MultivariateNormalDistribution ./ cl;

filler = zeros(512,512); %filler gets the val for the chanse it have at the current run of c
outimg = zeros(512,512);
for c = 1:cl
    for m = 1:512
        for n = 1:512
            if chance(c, m, n) > filler(m, n)
                filler(m,n) = chance(c, m, n);
                outimg(m,n) = c;
            end
        end
    end
end


confusion = confusionmat(outimg(:), actual_matrix(:)); %make the confusionmatrix
acc = sum(sum(diag(confusion)))/ sum(sum(confusion)) *100 ; %calc percent


% used For PLOTTING DURING REPORT. INGNORE
% figure();clf
% imagesc(squeeze(chance(1,:,:)))
% colormap hot
% title(' probability of class 1');
% figure();clf
% imagesc(squeeze(chance(2,:,:)))
% colormap hot
% title(' probability of class 2');
% figure();clf
% imagesc(squeeze(chance(3,:,:)))
% colormap hot
% title(' probability of class 3');
```

```
% figure();clf
% imagesc(squeeze(chance(4,:,:)))
% colormap hot
% title(' probability of class 4');

end
```