IN5520 – Digital Image Analysis

Mandatory 2

Paul Arquier – paularq@uio.no

Description of the problem

The goal of this assignment if to implement multivariate Gaussian classifier and use it to classify a set of images with 4 diffrent texture classes. The GLCMs and the features used are chosen by manual analysis prior to implementing and training the classifier.

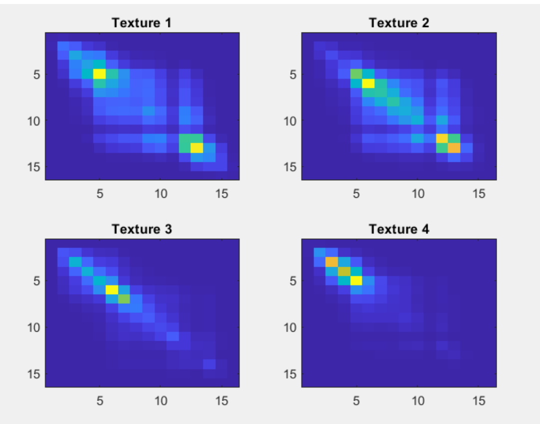
1. Choosing GLCM images to work with

In this part we want to find the 2 angles that will allow us to best differentiate the textures.

In figures 1 to 4 below, we can see some distinctions between the textures depending on the angle chosen.

We can clearly see that the 0 angle differentiates between textures 1 and 2 as well as textures 3 and 4. Angle 90 helps us to distinguish textures 2 and 3.

That is why we will choose angles 0 and 90.



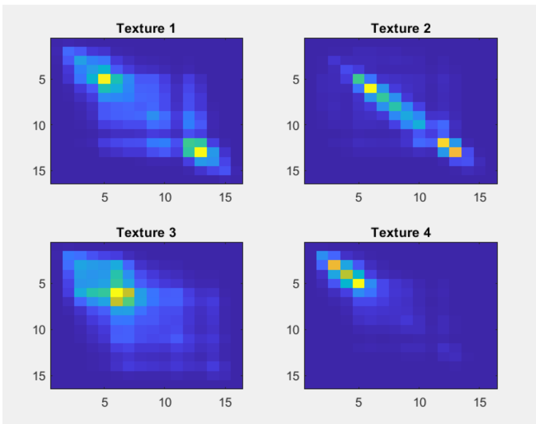
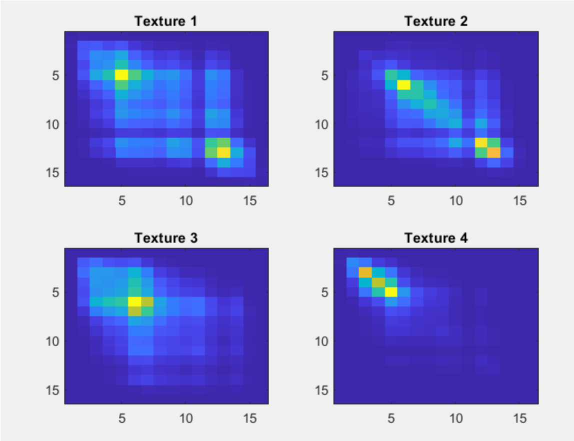


Figure 1 : Angle 90; dx=0 dy=-1 Figure 2 : Angle 0; dx=1 dy=0



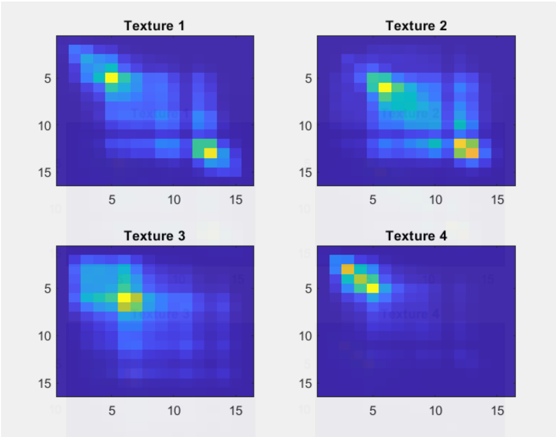


Figure 3 : Angle 45; dx=1 dy=-1 Figure 4 : Angle 135; dx=-1 dy=-1

2. Discussing new features by subdividing the GLCM matrices

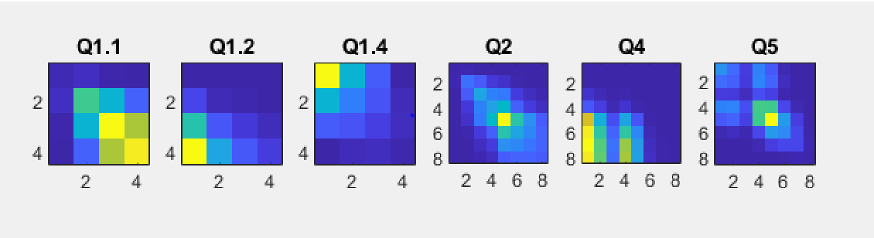
Here we have a vector Q = [Q1, Q2, Q3, Q4]where the values Q2 and Q3 will have the same value since the matrix is symmetric.

The local distribution of GLCM values is ignored giving the summed average. We can therefore obtain similar values for 2 different textures. To counter this problem, we will divide Q1 into 4 sub-quadrants Q1 = [Q1.1, Q1.2, Q1.3, Q1.4]. Q1.2 and Q1.3 will still have the same value which we will calculate once.

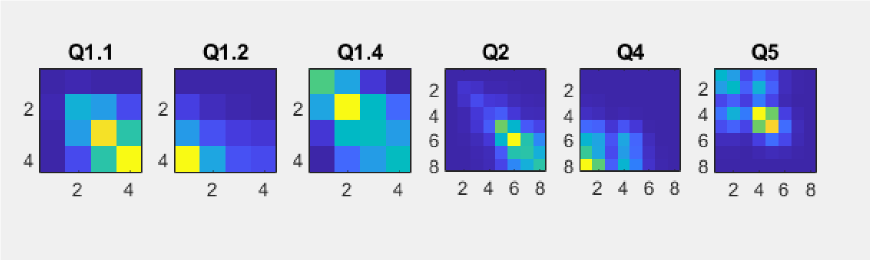
$$Q_{1.1} = \frac{\sum_{i=1}^{4}\sum_{j=1}^{4} P(i,j)}{\sum_{i=1}^{8}\sum_{j=1}^{8} P(i,j)}$$

$$Q_{1.2} = \frac{\sum_{i=5}^{8}\sum_{j=1}^{4} P(i,j)}{\sum_{i=1}^{8}\sum_{j=1}^{8} P(i,j)}$$

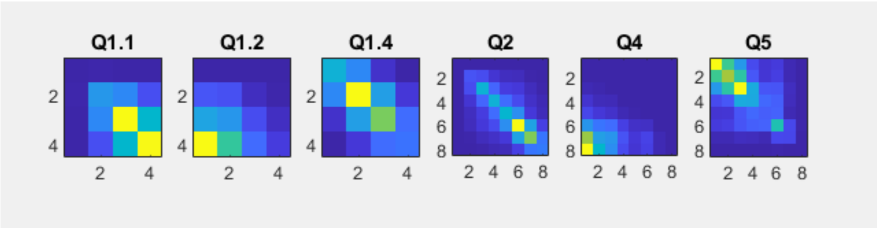
$$Q_{1.4} = \frac{\sum_{i=5}^{8}\sum_{j=5}^{8} P(i,j)}{\sum_{i=1}^{8}\sum_{j=1}^{8} P(i,j)}$$



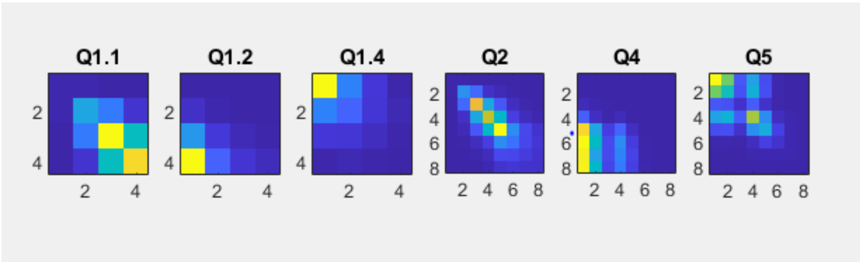
Texture 1



Texture 2



Texture 3



Texture 4

Figure 5 : Visualization of quadrants : angle 0, dx=1 dy=0

Q1.1: We can see a highest average value for texture 1.

Q1.2: Higher value for texture 3 and lower for texture 4.

Q1.4: Higher value for texture 2 and lower for texture 4.

Q2 : Here we can differentiate between textures 1 and 4 compared to textures 2 and 3.

Q4 : Here we can differentiate between textures 1 and 2 compared to textures 3 and 4.

With this analysis, we can focus on Q = [Q1.1, Q1.2, Q1.4, Q2, Q4] to analyse the textures.

3. Selecting and implementing a subset of these features

Here we will use the 0 and 90 angles as seen in the previous sections to analyse the features.

We can directly observe that there are many similarities among the quadrants and that it will be necessary to sort and focus on those that best differentiate the textures.

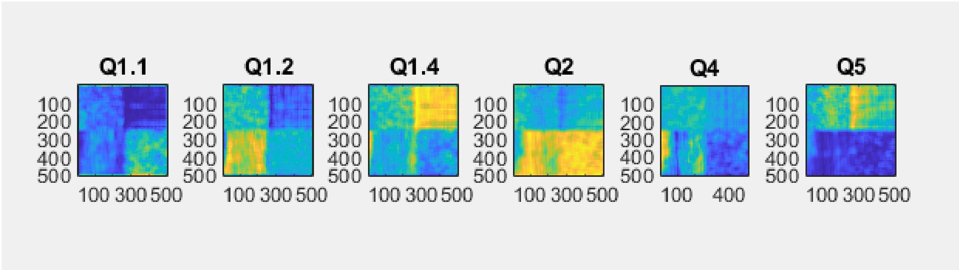


Figure 5 : feature images for 0 degree angle

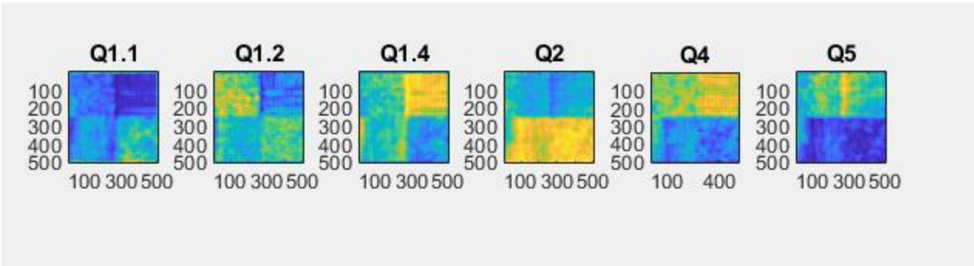


Figure 6 : feature images for 90 degree angle

Q2 and Q4 separate textures 1 and 2 from the others for angle 90. We can differentiate texture 4 with angle 0 and Q1.1 and Q2. Q1.1, Q1.2 and Q1.4 separate texture 2 from the others. Q1.2 with angle 0 also separates texture 3 from the others.

In the end, we will therefore choose the following feature : Q = [$Q_{1.2}(0\ degree)$, [$Q_{1.4}(90\ degree)$, [$Q_2(0\ degree)$, [$Q_2(90\ degree)$].

4. Implement a multivariate Gaussian classifier

Multivariate gaussian classifier is implemented using 3 functions : gaussianClassifier, gaussianTrainer and gaussianEvaluator as we can see in the code below.

```matlab
function [class] = gaussianClassifier(feats, labels, means, covs)

[N, M] = size(feats{1});
class = zeros(N, M);
feat_count = numel(feats);
class_count = numel(labels);

for n = 1:N
    for m = 1:M
        G_count = zeros(feat_count, 1);

        for i = 1:feat_count
            window = feats{i};
            G = double(window(n, m));
            G_count(i) = G;
        end

        max_class = 0;
        max_val = 0;

        for i = 1:class_count
            cov = covs(:, :, i);

            gauss = - (1/2)*(G_count - means(:, i))'*inv(cov)* ...
                (G_count - means(:, i)) - (feat_count/2)*log(2*pi) - ...
                (1/2)*log(det(cov)) + log(1/class_count);

            if i == 1 || gauss > max_val
                max_class = labels(i);
                max_val = gauss;
            end
        end
        class(n, m) = max_class;
    end
end
end
```

```matlab
function [acc, avg_acc, conf] = gaussianEvaluator(class, class_count)

% Buffers for resulting matrices
acc = zeros(1, class_count);
conf = zeros(class_count);

[N, M] = size(class);

% Calculate the confusion matrix and accuracy counts
for n = 1:N
    for m = 1:M
        if n <= N/2
            if m <= M/2
                conf(1, class(n, m)) = conf(1, class(n, m)) + 1;
                if class(n, m) == 1
                    acc(1) = acc(1) + 1;
                end
            else
                conf(2, class(n, m)) = conf(2, class(n, m)) + 1;
                if class(n, m) == 2
                    acc(2) = acc(2) + 1;
                end
            end
        else
            if m <= M/2
                conf(3, class(n, m)) = conf(3, class(n, m)) + 1;
                if class(n, m) == 3
                    acc(3) = acc(3) + 1;
                end
            else
                conf(4, class(n, m)) = conf(4, class(n, m)) + 1;
                if class(n, m) == 4
                    acc(4) = acc(4) + 1;
                end
            end
        end
    end
end

% Calculate accuracy percentage and average
acc = acc./(N/2)^2;
avg_acc = mean(acc);
end
```

```
function [labels, means, covs] = gaussianTrainer(feats, train_mask)

% Buffers for resulting matrices
labels = unique(train_mask(train_mask > 0));
label_count = numel(labels);
feat_count = numel(feats);
means = zeros(feat_count, label_count);
covs = zeros(feat_count, feat_count, label_count);

% For each label
for i = 1:label_count
    mask = (train_mask == labels(i));

    % Calculate means
    means_tmp = zeros(feat_count, 1);
    feats_masked = [];
    for j=1:feat_count
        masked_img = feats{j}(mask == true);
        [n, m] = size(masked_img);
        means_tmp(j) = sum(masked_img(:)) / (n*m);
        feats_masked = [feats_masked masked_img];
    end
    means(:, i) = means_tmp;

    % Calculate covariances
    covs(:, :, i) = cov(double(feats_masked));
end
end
```

The posterior probability is calculated in gaussianClassifier.m using the Bayes rule with expression :

$$g_i(x) = -\frac{1}{2}(x-\mu_i)^T \sum^{-1}(x-\mu_i) - \frac{d}{2}\log 2\pi - \frac{1}{2}\log\left|\sum{}_i\right| + \log p(w_i)$$

In gaussianTrainer, we calculate the mean vector, μ and the covariance matrix Σ.

We can use the value calculate in gaussianClassifier in the gaussianEvalator function to estimate the accuracy of the classification and the confusion matrix.

5. Training the classifier based on the feature subset from point 3

Below in figure 7, we can see the result of the classification.

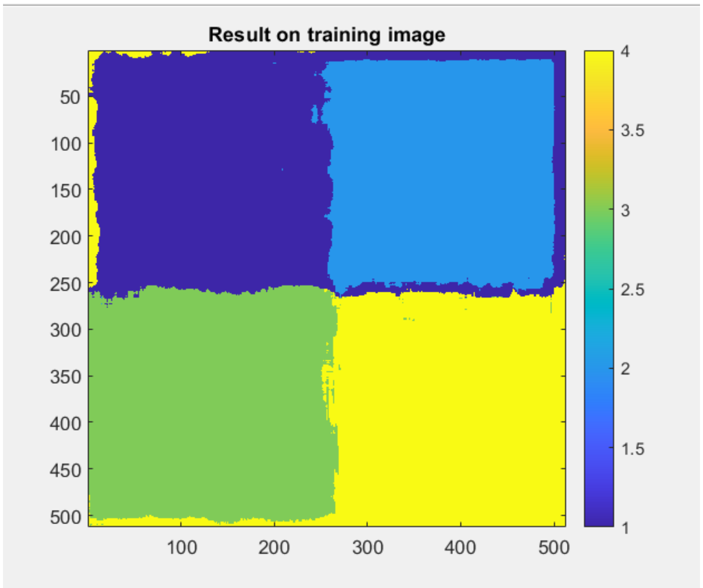


Figure 7 : Result of the classification of the training image

```
acc =

    0.9548    0.8835    0.9531    0.9425


avg_acc =

    0.9335


conf =

       62573         333         204        2426
        7605       57902           0          29
         948           4       62465        2119
        1850          80        1838       61768
```

The overall classification accuracy is approximately 93%.

The worst accuracy is 88% for the texture 2.

The result is not bad but the zero padding seems to influence the final result. We can see it on the picture with some imperfections.

6. Evaluation of classification performance on the test data set using the set of features selected in point 3

In this part we can see the difference for the classification for image 1 (figure 8) and image 2 (figure 9) when we use corresponding mask or the original.

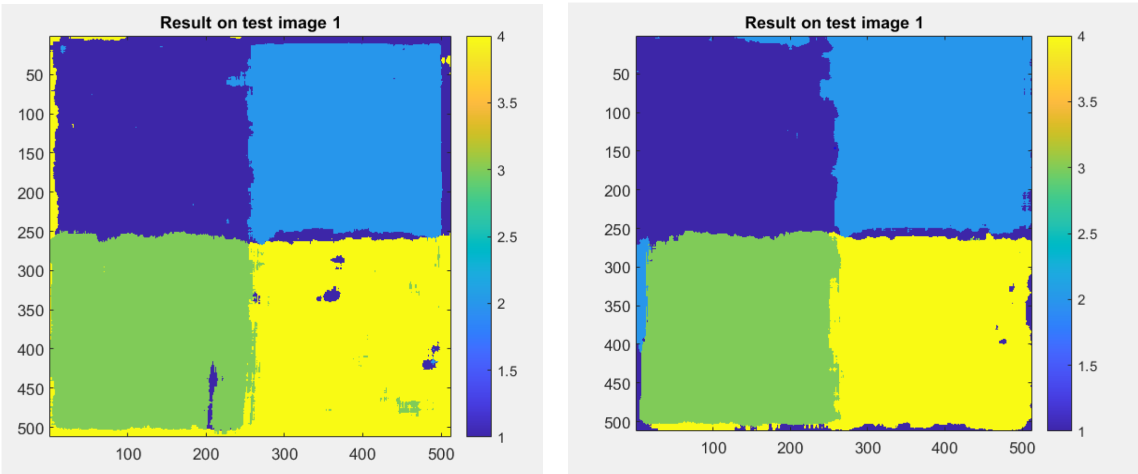


Figure 8 : Classification result of the test image 1 vs classification result of the test image 1 with corresponding mask

```
acc1 =

    0.9449    0.8932    0.9324    0.9384


avg_acc1 =

    0.9272


conf1 =

       61926         803         574        2233
        6898       58540           0          98
        1044           5       61104        3383
        2210         216        1609       61501
```

```
acc1 =

    0.9698    0.9725    0.9040    0.9342


avg_acc1 =

    0.9451


conf1 =

       63557        1790         180           9
        1799       63733           0           4
        2245        2094       59245        1952
        2931         571         813       61221
```

For image 1, the average accuracy is 92.7% with the original mask versus 94.5% with the corresponding mask. So we have a slight increase of about 2%, which is relatively small.

In figure 8 we can see that the differences between the 2 images are that there is less inaccuracy with the corresponding mask.

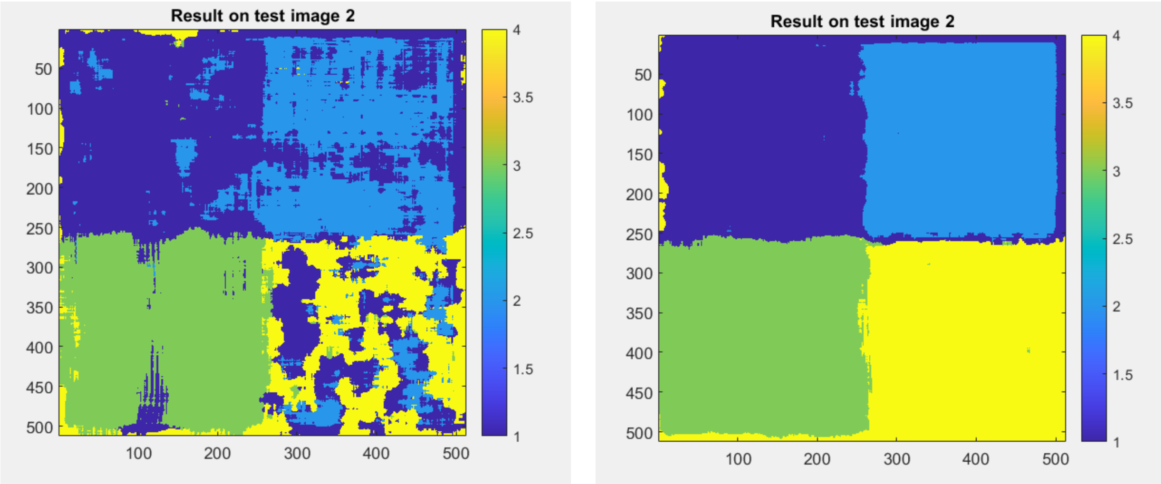


Figure 9 : Classification result of the test image 2 vs classification result of the test image 2with corresponding mask

```
acc2 =

    0.8962    0.6833    0.8936    0.4996


avg_acc2 =

    0.7432


conf2 =

       58735        5153         252        1396
       20548       44784           0         204
        4483          66       58563        2424
       21965        8917        1915       32739
```

```
acc2 =

    0.9781    0.8936    0.9524    0.9475


avg_acc2 =

    0.9429


conf2 =

       64102         266         241         927
        6968       58563           2           3
         871           0       62414        2251
        1660          44        1737       62095
```

For image 2, the average accuracy is 74.3% with the original mask versus 94.3% with the corresponding mask. There is a strong increase of about 20%.

Textures 2 and 4 have the most changes. For textures 1 and 3, the corresponding mask has removed the inaccuracies.

In this part, we can conclude that using corresponding mask allows us to obtain a more accurate result than 94%. In the first case, it allows to remove some inaccuracies, and in the second case it allows to really highlight the differences between the textures.

Conclusion

Throughout this exercise we have seen that by choosing the right parameters we can achieve good results in differentiating textures. Nevertheless, as we have seen in the previous section, using the original corresponding mask does not give a convincing result (figure 9). We have seen that we can solve this problem by using the corresponding mask.

In the end, the classifier seems to work well and be quite reliable if we use the right features.

Code

Mandatory2.m

```
close all;

% Choosing GLCM images to work with

texture1_90 = load('texture1dx0dymin1.txt');
texture2_90 = load('texture2dx0dymin1.txt');
texture3_90 = load('texture3dx0dymin1.txt');
texture4_90 = load('texture4dx0dymin1.txt');

figure(1)
subplot(221); imagesc(texture1_90); title('Texture 1');
subplot(222); imagesc(texture2_90); title('Texture 2');
subplot(223); imagesc(texture3_90); title('Texture 3');
subplot(224); imagesc(texture4_90); title('Texture 4');

texture1_0 = load('texture1dx1dy0.txt');
texture2_0 = load('texture2dx1dy0.txt');
texture3_0 = load('texture3dx1dy0.txt');
texture4_0 = load('texture4dx1dy0.txt');

figure(2)
subplot(221); imagesc(texture1_0); title('Texture 1');
subplot(222); imagesc(texture2_0); title('Texture 2');
subplot(223); imagesc(texture3_0); title('Texture 3');
subplot(224); imagesc(texture4_0); title('Texture 4');

texture1_45 = load('texture1dx1dymin1.txt');
texture2_45 = load('texture2dx1dymin1.txt');
texture3_45 = load('texture3dx1dymin1.txt');
texture4_45 = load('texture4dx1dymin1.txt');

figure(3)
subplot(221); imagesc(texture1_45); title('Texture 1');
subplot(222); imagesc(texture2_45); title('Texture 2');
subplot(223); imagesc(texture3_45); title('Texture 3');
subplot(224); imagesc(texture4_45); title('Texture 4');

texture1_135 = load('texture1dxmin1dymin1.txt');
texture2_135 = load('texture2dxmin1dymin1.txt');
texture3_135 = load('texture3dxmin1dymin1.txt');
texture4_135 = load('texture4dxmin1dymin1.txt');

```

```
figure(4)
subplot(221); imagesc(texture1_135); title('Texture 1');
subplot(222); imagesc(texture2_135); title('Texture 2');
subplot(223); imagesc(texture3_135); title('Texture 3');
subplot(224); imagesc(texture4_135); title('Texture 4');


% Discussing new features by subdividing the GLCM matrices

% Quadrant 1.1
t1_90_q11 = texture1_90(1:4, 1:4);
t2_90_q11 = texture2_90(1:4, 1:4);
t3_90_q11 = texture3_90(1:4, 1:4);
t4_90_q11 = texture4_90(1:4, 1:4);

% Quadrant 1.2
t1_90_q12 = texture1_90(1:4, 5:8);
t2_90_q12 = texture2_90(1:4, 5:8);
t3_90_q12 = texture3_90(1:4, 5:8);
t4_90_q12 = texture4_90(1:4, 5:8);

% Quadrant 1.4
t1_90_q14 = texture1_90(5:8, 5:8);
t2_90_q14 = texture2_90(5:8, 5:8);
t3_90_q14 = texture3_90(5:8, 5:8);
t4_90_q14 = texture4_90(5:8, 5:8);

% Quadrant 2

t1_90_q2 = texture1_90(1:8, 1:8);
t2_90_q2 = texture2_90(1:8, 1:8);
t3_90_q2 = texture3_90(1:8, 1:8);
t4_90_q2 = texture4_90(1:8, 1:8);

% Quadrant 4

t1_90_q4 = texture1_90(1:8, 9:16);
t2_90_q4 = texture2_90(1:8, 9:16);
t3_90_q4 = texture3_90(1:8, 9:16);
t4_90_q4 = texture4_90(1:8, 9:16);
```

```
% Quadrant 5
t1_90_q5 = texture1_90(9:16, 9:16);
t2_90_q5 = texture2_90(9:16, 9:16);
t3_90_q5 = texture3_90(9:16, 9:16);
t4_90_q5 = texture4_90(9:16, 9:16);

figure(5)
subplot(161); imagesc(t1_90_q11); title('Q1.1'); axis('square');
subplot(162); imagesc(t1_90_q12); title('Q1.2'); axis('square');
subplot(163); imagesc(t1_90_q14); title('Q1.4'); axis('square');
subplot(164); imagesc(t1_90_q2); title('Q2'); axis('square');
subplot(165); imagesc(t1_90_q4); title('Q4'); axis('square');
subplot(166); imagesc(t1_90_q5); title('Q5'); axis('square');

figure(6)
subplot(161); imagesc(t2_90_q11); title('Q1.1'); axis('square');
subplot(162); imagesc(t2_90_q12); title('Q1.2'); axis('square');
subplot(163); imagesc(t2_90_q14); title('Q1.4'); axis('square');
subplot(164); imagesc(t2_90_q2); title('Q2'); axis('square');
subplot(165); imagesc(t2_90_q4); title('Q4'); axis('square');
subplot(166); imagesc(t2_90_q5); title('Q5'); axis('square');

figure(7)
subplot(161); imagesc(t3_90_q11); title('Q1.1'); axis('square');
subplot(162); imagesc(t3_90_q12); title('Q1.2'); axis('square');
subplot(163); imagesc(t3_90_q14); title('Q1.4'); axis('square');
subplot(164); imagesc(t3_90_q2); title('Q2'); axis('square');
subplot(165); imagesc(t3_90_q4); title('Q4'); axis('square');
subplot(166); imagesc(t3_90_q5); title('Q5'); axis('square');

figure(8)
subplot(161); imagesc(t4_90_q11); title('Q1.1'); axis('square');
subplot(162); imagesc(t4_90_q12); title('Q1.2'); axis('square');
subplot(163); imagesc(t4_90_q14); title('Q1.4'); axis('square');
subplot(164); imagesc(t4_90_q2); title('Q2'); axis('square');
subplot(165); imagesc(t4_90_q4); title('Q4'); axis('square');
subplot(166); imagesc(t4_90_q5); title('Q5'); axis('square');

% Selecting and implementing a subset of these features
```

```
train_img = load('mosaic1_train.txt');

% Quantizing to G gray levels
G = 16;
train_img = uint8(round(double(train_img)*(G - 1)/double(max(train_img(:)))));

% Getting the feature images
windowSize = 31;
[Q1_1, Q1_2, Q1_4, Q2, Q4, Q5] = glidingGLCM(train_img, G, 1, 0, windowSize, 0);
[K1_1, K1_2, K1_4, K2, K4, K5] = glidingGLCM(train_img, G, 1, 90, windowSize, 0);

figure(9)
subplot(161); imagesc(Q1_1); title('Q1.1'); axis('square');
subplot(162); imagesc(Q1_2); title('Q1.2'); axis('square');
subplot(163); imagesc(Q1_4); title('Q1.4'); axis('square');
subplot(164); imagesc(Q2); title('Q2'); axis('square');
subplot(165); imagesc(Q4); title('Q4'); axis('square');
subplot(166); imagesc(Q5); title('Q5'); axis('square');
%suptitle('0 degree angle');

figure(10)
subplot(161); imagesc(K1_1); title('Q1.1'); axis('square');
subplot(162); imagesc(K1_2); title('Q1.2'); axis('square');
subplot(163); imagesc(K1_4); title('Q1.4'); axis('square');
subplot(164); imagesc(K2); title('Q2'); axis('square');
subplot(165); imagesc(K4); title('Q4'); axis('square');
subplot(166); imagesc(K5); title('Q5'); axis('square');
%suptitle('90 degree angle');


% Training the classifier based on the feature subset from point 3

train_img = load('mosaic1_train.txt');

% Quantizing to G gray levels
G = 16;
train_img = uint8(round(double(train_img)*(G - 1)/double(max(train_img(:)))));

windowSize = 31;
[Q1_1, Q1_2, Q1_4, Q2, Q4, Q5] = glidingGLCM(train_img, G, 1, 0, windowSize, 0);
```

```
[K1_1, K1_2, K1_4, K2, K4, K5] = glidingGLCM(train_img, G, 1, 90, windowSize, 0)


feats = {Q1_2, K1_4, Q2, K2};

% Using gaussianTrainer
train_mask = load('training_mask.txt');
[labels, means, covs] = gaussianTrainer(feats, train_mask);

% Using gaussianClassifier
[class] = gaussianClassifier(feats, labels, means, covs);

% Using gaussianEvaluator
[acc, avg_acc, conf] = gaussianEvaluator(class, 4)

figure(11)
imagesc(class); colorbar; title('Result on training image'); axis('square');

test_img1 = load('mosaic2_test.txt');
test_img2 = load('mosaic3_test.txt');

G = 16;
test_img1 = uint8(round(double(test_img1)*(G - 1)/double(max(test_img1(:)))));
test_img2 = uint8(round(double(test_img2)*(G - 1)/double(max(test_img2(:)))));

windowSize = 31;
[Q1_1, Q1_2, Q1_4, Q2, Q4] = glidingGLCM(test_img1, G, 1, 0, windowSize, 0);
[K1_1, K1_2, K1_4, K2, K4] = glidingGLCM(test_img1, G, 1, 90, windowSize, 0);
feats1 = {Q1_2, K1_4, Q2, K2};

[Q1_1, Q1_2, Q1_4, Q2, Q4] = glidingGLCM(test_img2, G, 1, 0, windowSize, 0);
[K1_1, K1_2, K1_4, K2, K4] = glidingGLCM(test_img2, G, 1, 90, windowSize, 0);
feats2 = {Q1_2, K1_4, Q2, K2};

% Using gaussianClassifier
[class1] = gaussianClassifier(feats1, labels, means, covs);
[class2] = gaussianClassifier(feats2, labels, means, covs);

% Using gaussianEvaluator
[acc1, avg_acc1, conf1] = gaussianEvaluator(class1, 4)
```

```
[acc2, avg_acc2, conf2] = gaussianEvaluator(class2, 4)

figure(12)
imagesc(class1); colorbar; title('Result on test image 1'); axis('square');
figure(13)
imagesc(class2); colorbar; title('Result on test image 2'); axis('square');


% Evaluation of classification performance on the test data set using the set of features
% selected in point 3

train_img = load('mosaic1_train.txt');
test_img1 = load('mosaic2_test.txt');
test_img2 = load('mosaic3_test.txt');

G = 16;
train_img = uint8(round(double(train_img)*(G - 1)/double(max(train_img(:)))));
test_img1 = uint8(round(double(test_img1)*(G - 1)/double(max(test_img1(:)))));
test_img2 = uint8(round(double(test_img2)*(G - 1)/double(max(test_img2(:)))));

windowSize = 31;
[Q1_1, Q1_2, Q1_4, Q2, Q4, Q5] = glidingGLCM(train_img, G, 1, 0, windowSize, 0);
[K1_1, K1_2, K1_4, K2, K4, K5] = glidingGLCM(train_img, G, 1, 90, windowSize, 0);
feats = {Q1_2, K1_4, Q2, K2};

[Q1_1, Q1_2, Q1_4, Q2, Q4, Q5] = glidingGLCM(test_img1, G, 1, 0, windowSize, 0);
[K1_1, K1_2, K1_4, K2, K4, K5] = glidingGLCM(test_img1, G, 1, 90, windowSize, 0);
feats1 = {Q1_2, K1_4, Q2, K2};

[Q1_1, Q1_2, Q1_4, Q2, Q4, Q5] = glidingGLCM(test_img2, G, 1, 0, windowSize, 0);
[K1_1, K1_2, K1_4, K2, K4, K5] = glidingGLCM(test_img2, G, 1, 90, windowSize, 0);
feats2 = {Q1_2, K1_4, Q2, K2};

% Using gaussiantTrainer
train_mask2 = load('mask_mosaic2_test.mat');
train_mask2 = cell2mat(struct2cell(train_mask2));
train_mask3 = load('mask_mosaic3_test.mat');
train_mask3 = cell2mat(struct2cell(train_mask3));
[labels2, means2, covs2] = gaussianTrainer(feats, train_mask2);
[labels3, means3, covs3] = gaussianTrainer(feats, train_mask3);

% Using gaussianClassifier
[class1] = gaussianClassifier(feats, labels2, means2, covs2);
[class2] = gaussianClassifier(feats, labels3, means3, covs3);

% Using gaussianEvaluator
[acc1, avg_acc1, conf1] = gaussianEvaluator(class1, 4)
[acc2, avg_acc2, conf2] = gaussianEvaluator(class2, 4)

figure(14)
imagesc(class1); colorbar; title('Result on test image 1'); axis('square');
figure(15)
imagesc(class2); colorbar; title('Result on test image 2'); axis('square');
```

glidingGLCM.m

```
function [Q1_1, Q1_2, Q1_4, Q2, Q4, Q5] = glidingGLCM(window, grayscale, d, theta, windowSize, iso)
% Calculate the GLCM for every glading window in an image

[MOriginal, NOriginal] = size(window); % Original image size
HalfSize = floor(windowSize/2); % Size of half the filter

% Apply the zero-padding to the original image
padded = zeros(MOriginal + windowSize - 1, NOriginal + windowSize - 1);
padded(HalfSize:end - HalfSize - 1, HalfSize:end - HalfSize - 1) = window;

[M, N] = size(padded); % Padded image size

% Buffers for resulting images
Q1_1 = zeros(MOriginal, NOriginal);
Q1_2 = zeros(MOriginal, NOriginal);
Q1_4 = zeros(MOriginal, NOriginal);
Q2 = zeros(MOriginal, NOriginal);
Q4 = zeros(MOriginal, NOriginal);
Q5 = zeros(MOriginal, NOriginal);

% Go through the image
for m = (HalfSize + 1):(M - HalfSize - 1)
    for n = (HalfSize + 1):(N - HalfSize - 1)

        % Extracting the window
        window = padded(m - HalfSize:m + HalfSize, ...
            n - HalfSize:n + HalfSize);

        % Calculating the GLCM
        if iso == 1
            p = isoGLCM(window, grayscale, d);
        else
            p = GLCM(window, grayscale, d, theta);
        end

        % Calculating the features
        Q1_1(m - HalfSize, n - HalfSize) = sum(sum(p(1:4, 1:4)))/sum(sum(p(1:8, 1:8)));
        Q1_2(m - HalfSize, n - HalfSize) = sum(sum(p(1:4, 5:8)))/sum(sum(p(1:8, 1:8)));
        Q1_4(m - HalfSize, n - HalfSize) = sum(sum(p(5:8, 5:8)))/sum(sum(p(1:8, 1:8)));
        Q2(m - HalfSize, n - HalfSize) = sum(sum(p(1:8, 1:8)))/sum(sum(p));
        Q4(m - HalfSize, n - HalfSize) = sum(sum(p(1:8, 9:16)))/sum(sum(p));
        Q5(m - HalfSize, n - HalfSize) = sum(sum(p(9:16, 9:16)))/sum(sum(p));

    end
end
end
```

GLCM.m

```matlab
function [glcm] = GLCM(window, grayscale, d, theta)
% GLCM function calculates the GLCM of an image and the result is
% normalized and symmetric

[N,M] = size(window);
glcm = zeros(grayscale);

% Translating input
if theta == 0
    dx = d;
    dy = 0;
elseif theta == 45
    dx = d;
    dy = d;
elseif theta == 90
    dx = 0;
    dy = d;
elseif theta == -45
    dx = d;
    dy = d;
    window = flipud(window);
end

% Counting transitions for indexing
for i = 1:N
    for j = 1:M
        if i + dy > N || i + dy < 1 || i + dx < 1 || ...
           j + dx > M || j + dy < 1 || j + dx < 1
           continue
        end
        first = window(i,j);
        second = window(i + dy, j + dx);
        glcm(first + 1, second + 1) = glcm(first + 1, second + 1) + 1;
    end
end

% Making symmetric and normalize
glcm = glcm + glcm';
glcm = glcm/sum(sum(glcm));
end
```