IN5520 - Mandatory Assignment 2

Ada Weinert Ravn, adalw@ulrik.uio.no

November 12, 2019

1 Introduction

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

2 Choosing GLCM images to work with

The GLCM image(s) need to be chosen in such a way that each texture can be clearly distinguished from the others.

From the first look the GLCMs of the 0 degree direction are the only ones showing a clear distinction between Textures 1 and 2, making it a necessary inclusion. The distinction between Textures 3 and 4 also appears to be the best we can get for this set. The distinction between Textures 1 and 4 is good, although notably the one for the 45 degree direction is more defined.

This leaves distinctions between Textures 1 and 3, as well as Textures 2 and 4. Based on the peak value distribution the GLCMs of the 0 degree direction should provide a good enough distinction, though this can be supplemented by the GLCMs of the 90 degree direction which have clearer distinction between the two sets of textures.

The chosen GLCMs can be seen in Figure 1 and 2, and the code used to visualize the GLCMs under Mandatory2_1.m in Appendix.

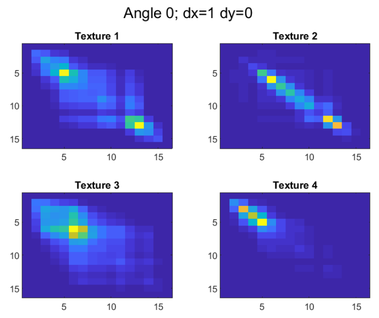

Figure 1: GLCMs of angle 0, dx=1, dy=0.

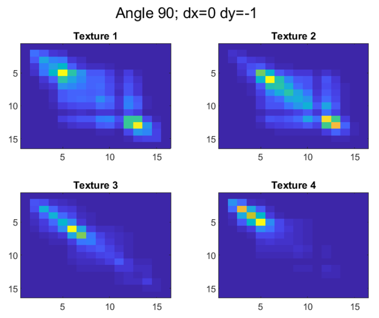

Figure 2: GLCMs of angle 90, dx=0, dy=-1.

3 Discussing new features by subdividing the GLCM matrices

The task proposes the feature vector $Q = [Q1, Q2, Q3, Q4]$ representing the percentage of gray levels found in each quadrant. This will however result in Q2 and Q3 having the same value since the GLCMs are diagonally mirrored. The proposed feature also ignores the local distribution of GLCM values by giving the summed average.

To try to combat this Q1 will be subdivided into four subquadrants $Q1 = [Q1.1, Q1.2, Q1.3, Q1.4]$, using the formula $Q1.1 = \frac{\sum_{i=1}^{4}\sum_{j=1}^{4} P(i,j)}{\sum_{i=1}^{8}\sum_{j=1}^{8} P(i,j)}$, adjusted for each subquadrant. Once again the value of Q1.2 and Q1.3 will be the same. This quadrant has been chosen since it seems to store the most information across the different textures.

Based on this analysis we will inspect he following quadrants for the GLCMs in Figure 1: Q1.1, Q1.2, Q1.4, Q2 and Q4. The visualisations can be seen in Figure 3, and the code used under Mandatory2_2.m in Appendix.

Since the feature will be the summed average of the GLCM values the analysis of the most useful ones will be slightly difficult based on images alone and not the actual values. Some prediction can be made though.

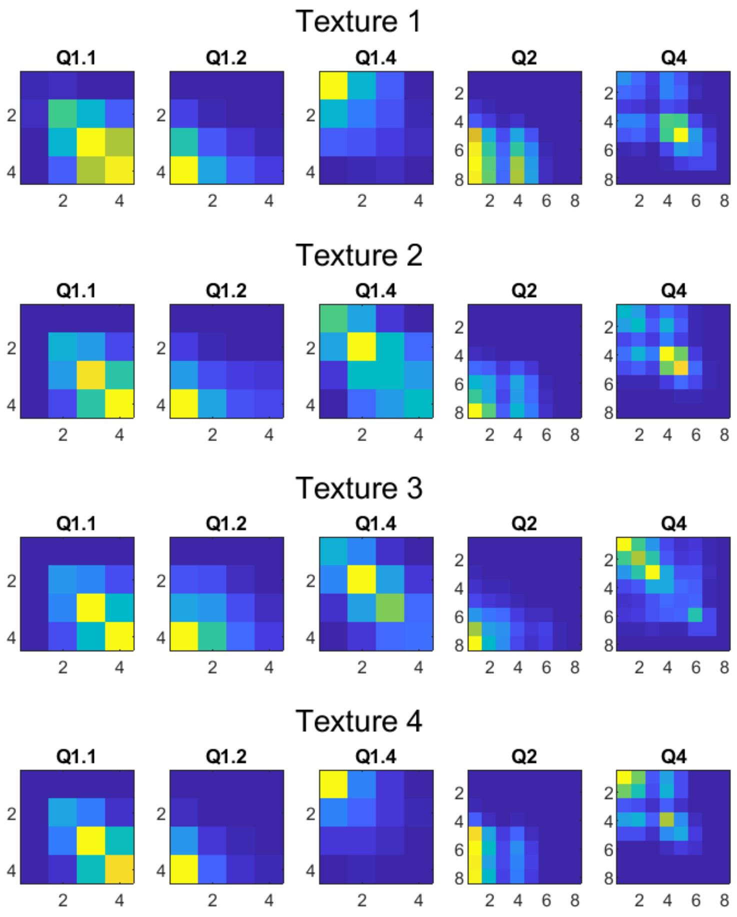

Figure 3: Quadrant visualization of angle 0, dx=1, dy=0.

Q1.1: Texture 1 seems to be distinguishable from the others with the highest average value.
Q1.2: Slightly distinguishable Textures 3 and 4 with highest and lowest values respectively.
Q1.4: Textures 2 and 4 are clearly distinguishable with highest and lowest value respectively.
Q2 : Textures 1 and 4 are clearly distinguished from Textures 2 and 3.
Q4 : Textures 1 and 2 seem slightly distinguished from Textures 3 and 4.

Based on this analysis we will take a closer look at the following set of textures: $Q = [Q1.1, Q1.2, Q1.4, Q2, Q4]$. The obvious exclusions have been removed, and the expectation is that not all of the above will be needed.

4 Selecting and implementing the best features from the GLCM matrices

Taking into account the analysis from the previous parts the features have been calculated for both 0 and 90 degree angle GLCMs.

The code used to compute this task has been in part based on updated code from Mandatory Assignment 1, and is listed under Mandatory2_3.m, GLCM.m and glidingGLCM.m in Appendix. The produced can be seen in Figure 4 below.

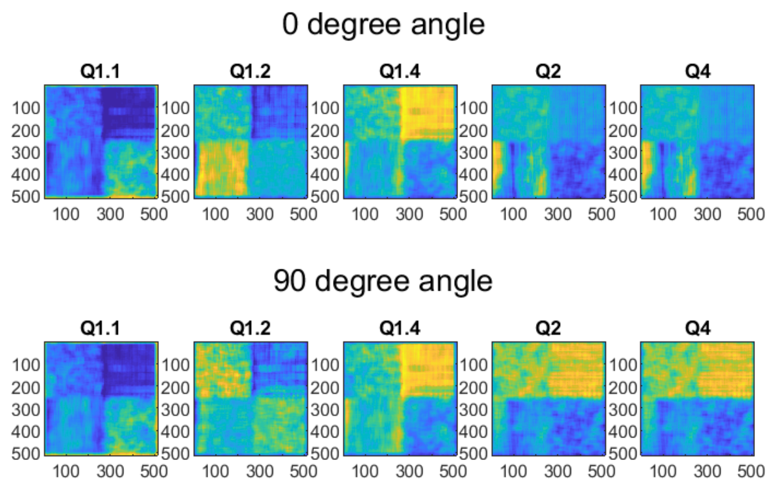

Figure 4: Feature images for the selected quadrants and angles.

From the brief look alone we can clearly see that not all of the above quadrants will be necessary. In case of both angles Q2 and Q4 are nearly identical, and Q1.1 and Q1.4 seem to be the opposites of each other with highest and lowest values switched.

The produced features for each angle are very similar with the exception of Q2 and Q4 which clearly split Textures 1 and 2 from Textures 3 and 4 for the 90 degree angle.

Q1.1 and Q2 (0 degree) seem to separate Texture 4 quite well and Q1.1, Q1.2 and Q1.4 distinguish Texture 2. Q1.2 also seems to distinguish Texture 2 and Texture 3 from the other ones by lowest and highest values respectively.

Based on this analysis we will be using the following feature set:
$Q = [Q1.2_{0degree}, Q1.4_{90degree}, Q2_{0degree}, Q2_{90degree}]$

5 Implementing a Gaussian classifier

The multivariate Gaussian classifier is implemented as a set of three functions: multivatiateGaussianClassifier.m, multivatiateGaussianTrainer.m and multivatiateGaussianEvaluator.m.

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

$$g_i(x) = -\frac{1}{2}(x-\mu_i)^T\Sigma^{-1}(x-\mu_i) - \frac{d}{2}log2\pi - \frac{1}{2}log|\Sigma_i| + logp(\omega_i)$$

The calculation of mean vector, μ and covariance matrix Σ is done in multivatiateGaussianTrainer.m using built in MATLAB functions mean and cov respectively. The trainer needs to be called first using the mask provided and calculated feature matrices stored in cells.

The classification returned by multivatiateGaussianClassifier can be then used in multivatiateGaussianEvaluaton to estimate the accuracy of the classification and the confusion matrix.

The code for each of the functions is listed below.

```
function [class] = multivatiateGaussianClassifier(feats, labels, ...
    means, covs)

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

for n = 1:N
    for m = 1:M
        G_cnt = zeros(feat_cnt, 1);

        for i = 1:feat_cnt
            img = feats{i};
            G = double(img(n, m));
            G_cnt(i) = G;
        end

        max_class = 0;
        max_val = 0;

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

            gauss = - (1/2)*(G_cnt - means(:, i))'*inv(cov)* ...
                (G_cnt - means(:, i)) - (feat_cnt/2)*log(2*pi) - ...
                (1/2)*log(det(cov)) + log(1/class_cnt);

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

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

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

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

    % Calculate means
    means_tmp = zeros(feat_cnt, 1);
    feats_masked = [];
    for j=1:feat_cnt
        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
```

```matlab
function [acc, avg_acc, conf] = ...
    multivatiateGaussianEvaluator(class, class_cnt)

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

[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
```

6 Training the classifier on the chosen images

The result of the classification can be seen in Figure 5, and the code used to visualize it and calculate results can be seen under Mandatory2_5.m in Appendix.

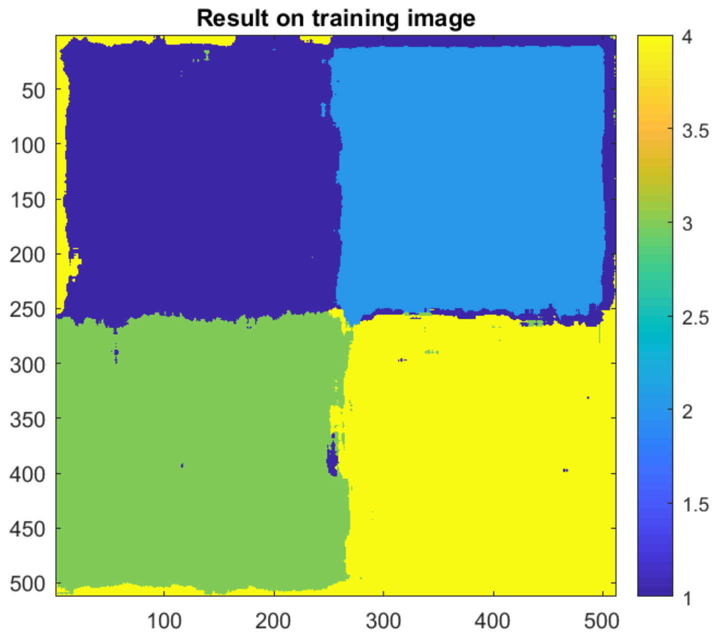

Figure 5: Classification result of the training image.

```
acc =
    0.9304    0.8886    0.9574    0.9470

avg_acc =
    0.9308

conf =
       60972         347         186        4031
        6725       58234         332         245
         991           0       62744        1801
         740         114        2619       62063
```

The overall classification accuracy is approximately 93% with Texture 2 having overall worst accuracy. The zero padding seems to significantly affect the result as seen in the image due to smudging, but small inaccuracies can be seen within the image as well. Overall this is not an ideal but acceptable result.

7 Classifying the test images

The code for the classification using the original mask can be seen under Mandatory_5.m and for the alternate masks under Mandatory2_6.m in Appendix.

The test images are slightly different from the train image causing varying degree of success. Those will be analysed below starting with the first test image.

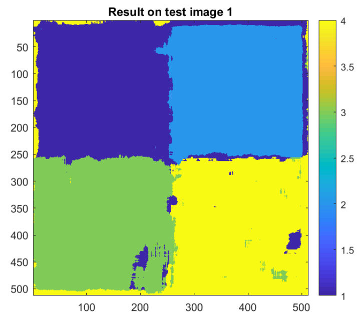

Figure 6: Classification result of the test image 1.

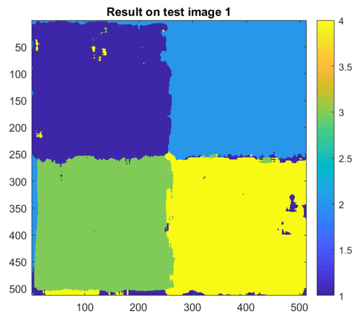

Figure 7: Classification result of the test image 1 with corresponding mask.

```
acc1 =
    0.9399    0.8984    0.9241    0.9379

avg_acc1 =
    0.9251

conf1 =
       61595          978          267         2696
        5911        58877          396          352
        2148            3        60564         2821
        1557          187         2329        61463

acc1v2 =
    0.9555    0.9814    0.9115    0.9513

avg_acc1v2 =
    0.9499

conf1v2 =
       62619         2167          340          410
         912        64316          107          201
        1980         2243        59737         1576
        1436          605         1151        62344
```

The results for test image 1 can be seen in Figures 6 and 7 as well as in the snippet above. In case of the original mask the overall accuracy is roughly the same, but with more inaccurate pockets within the image. The textures have been most commonly misclassified as Texture 1 as seen in the confusion matrix.

For corresponding mask the result is 2.5% higher than either result, showing significantly better results along the edges of the image. Notably the texture with the least accuracy in the first case scores the highest in the later case.

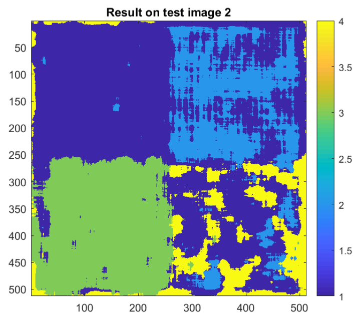

Figure 8: Classification result of the test image 2.

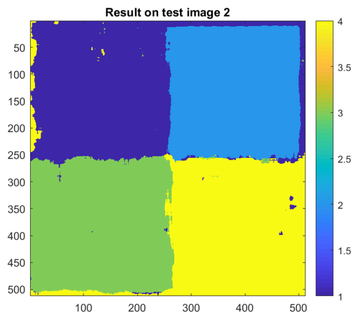

Figure 9: Classification result of the test image 2 with corresponding mask.

```
acc2 =
    0.9606    0.5568    0.8915    0.3316

avg_acc2 =
    0.6851

conf2 =
       62953         264         212        2107
       28538       36491         256         251
        5325           4       58425        1782
       35589        6653        1563       21731

acc2v2 =
    0.9601    0.9004    0.9538    0.9636

avg_acc2v2 =
    0.9445

conf2v2 =
       62919         247         312        2058
        6013       59010          85         428
         836           0       62510        2190
         581         116        1690       63149
```

The results for test image 1 can be seen in Figures 8 and 9 as well as in the snippet above. Here the result using original mask is quite bad scoring only 68% on accuracy. While Textures 1 and 3 remain quite alright compared to before, the remaining ones, and here especially Texture 4 look horrible. The Texture 4 has been in higher percentage misclassified as Texture 1 than correctly.

For corresponding mask the result is quite good, scoring approximately 94% accuracy. Small speck of misclassification can be seen, but errors appear mostly on the edges of the image.

8 Conclusion

The chosen feature set provides good results on train image and the first test image, but fails miserably at distinguishing textures in the second test image. Using dedicated mask for the test images resulted in good results in both cases using the same feature set.

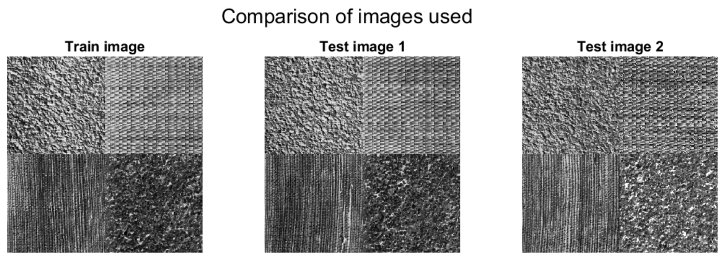

Figure 10: Images used.

By taking a look at the images used in Figure 10 we can consider the differences in performance for each image. For test image 1 the textures seem to be slightly offset, which not significantly influence the classification. The classifier is however unable to work with drastically different graylevel compared to the train image.

Overall the classifier seems to be working well and the issue seen in the second train image could be resolved by preprocessing, such as normalizing the images beforehand

9 Appendix

Mandatory2_1.m

```
clear all;
close all;

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');
suptitle('Angle 90; dx=0 dy=-1');


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');
suptitle('Angle 0; dx=1 dy=0');


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');
suptitle('Angle 45; dx=1 dy=-1');


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');
suptitle('Angle 135; dx=-1 dy=-1');
```

Mandatory2_2.m

```
clear all;
close all;

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

% 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, 9:16);
t2_90_q2 = texture2_90(1:8, 9:16);
t3_90_q2 = texture3_90(1:8, 9:16);
t4_90_q2 = texture4_90(1:8, 9:16);

% Quadrant 4
t1_90_q4 = texture1_90(9:16, 9:16);
t2_90_q4 = texture2_90(9:16, 9:16);
t3_90_q4 = texture3_90(9:16, 9:16);
t4_90_q4 = texture4_90(9:16, 9:16);

figure(1)
subplot(151); imagesc(t1_90_q11); title('Q1.1'); axis('square');
subplot(152); imagesc(t1_90_q12); title('Q1.2'); axis('square');
subplot(153); imagesc(t1_90_q14); title('Q1.4'); axis('square');
subplot(154); imagesc(t1_90_q2); title('Q2'); axis('square');
subplot(155); imagesc(t1_90_q4); title('Q4'); axis('square');
suptitle('Texture 1');

figure(2)
subplot(151); imagesc(t2_90_q11); title('Q1.1'); axis('square');
subplot(152); imagesc(t2_90_q12); title('Q1.2'); axis('square');
subplot(153); imagesc(t2_90_q14); title('Q1.4'); axis('square');
subplot(154); imagesc(t2_90_q2); title('Q2'); axis('square');
subplot(155); imagesc(t2_90_q4); title('Q4'); axis('square');
suptitle('Texture 2');

figure(3)
subplot(151); imagesc(t3_90_q11); title('Q1.1'); axis('square');
subplot(152); imagesc(t3_90_q12); title('Q1.2'); axis('square');
subplot(153); imagesc(t3_90_q14); title('Q1.4'); axis('square');
subplot(154); imagesc(t3_90_q2); title('Q2'); axis('square');
subplot(155); imagesc(t3_90_q4); title('Q4'); axis('square');
suptitle('Texture 3');

figure(4)
subplot(151); imagesc(t4_90_q11); title('Q1.1'); axis('square');
subplot(152); imagesc(t4_90_q12); title('Q1.2'); axis('square');
subplot(153); imagesc(t4_90_q14); title('Q1.4'); axis('square');
subplot(154); imagesc(t4_90_q2); title('Q2'); axis('square');
subplot(155); imagesc(t4_90_q4); title('Q4'); axis('square');
suptitle('Texture 4');
```

Mandatory2_3.m

```
clear all;
close all;

% Loading train image
train_img = load('mosaic1_train.txt');

% Quantizing to G gray levels
G = 16; % grayscale levels
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] = glidingGLCM(train_img, G, 1, 0, ...
    windowSize, 0);
[K1_1, K1_2, K1_4, K2, K4] = glidingGLCM(train_img, G, 1, 90, ...
    windowSize, 0);

% Visualizing the feature images
figure(1)
subplot(151); imagesc(Q1_1); title('Q1.1'); axis('square');
subplot(152); imagesc(Q1_2); title('Q1.2'); axis('square');
subplot(153); imagesc(Q1_4); title('Q1.4'); axis('square');
subplot(154); imagesc(Q2); title('Q2'); axis('square');
subplot(155); imagesc(Q4); title('Q4'); axis('square');
suptitle('0 degree angle');

figure(2)
subplot(151); imagesc(K1_1); title('Q1.1'); axis('square');
subplot(152); imagesc(K1_2); title('Q1.2'); axis('square');
subplot(153); imagesc(K1_4); title('Q1.4'); axis('square');
subplot(154); imagesc(K2); title('Q2'); axis('square');
subplot(155); imagesc(K4); title('Q4'); axis('square');
suptitle('90 degree angle');
```

GLCM.m

```
function [glcm] = GLCM(img, G, d, theta)
% GLCM calculates the GLCM (Gray Level Coocurrence Matrices) of ...
    an image.
% The result is normalized and symmetric.

[N,M] = size(img);
glcm = zeros(G);

% 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;
    img = flipud(img);
end

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

% Making symmetric
glcm = glcm + glcm';

% Normalizing
glcm = glcm/sum(sum(glcm));
end
```

gliding GLCM.m

```
function [Q1_1, Q1_2, Q1_4, Q2, Q4] = glidingGLCM(img, G, d, ...
    theta, w_s, iso)
% Calculate the feature of GLCM for every gliding window in an ...
    image. It
% adds a frame to the image first so that the resulting image is ...
    the same
% size as input.
% iso : 1 for isometric GLCM, 0 otherwise

[M_o, N_o] = size(img); % Original image size
h_s = floor(w_s/2); % Size of half the filter

% Apply the zero-padding to the original image
imgPadded = zeros(M_o + w_s - 1, N_o + w_s - 1);
imgPadded(h_s:end - h_s - 1, h_s:end - h_s - 1) = img;

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

% Buffers for resulting images
Q1_1 = zeros(M_o, N_o);
Q1_2 = zeros(M_o, N_o);
Q1_4 = zeros(M_o, N_o);
Q2 = zeros(M_o, N_o);
Q4 = zeros(M_o, N_o);

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

        % Extracting the window
        window = imgPadded(m - h_s:m + h_s, ...
            n - h_s:n + h_s);

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

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

Mandatory2_5.m

```
clear all;
close all;

% Loading train image
train_img = load('mosaic1_train.txt');

% Quantizing to G gray levels
G = 16; % grayscale levels
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] = glidingGLCM(train_img, G, 1, 0, ...
    windowSize, 0);
[K1_1, K1_2, K1_4, K2, K4] = glidingGLCM(train_img, G, 1, 90, ...
    windowSize, 0);

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

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

% Classification
[class] = multivatiateGaussianClassifier(feats, labels, means, ...
    covs);

% Evaluation
[acc, avg_acc, conf] = multivatiateGaussianEvaluator(class, 4)

% Visualization
figure(1)
imagesc(class); colorbar; title('Result on training image'); ...
    axis('square');


% Loading test image 1 and 2
test_img1 = load('mosaic2_test.txt');
test_img2 = load('mosaic3_test.txt');

% Quantizing to G gray levels
G = 16; % grayscale levels
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(:)))));

% Getting the feature images
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};

% Classification
[class1] = multivatiateGaussianClassifier(feats1, labels, means, ...
    covs);
[class2] = multivatiateGaussianClassifier(feats2, labels, means, ...
    covs);

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

% Visualization
figure(2)
imagesc(class1); colorbar; title('Result on test image 1'); ...
    axis('square');
figure(3)
imagesc(class2); colorbar; title('Result on test image 2'); ...
    axis('square');
```

Mandatory2_6.m

```
clear all;
close all;

% Loading train image
train_img = load('mosaic1_train.txt');
test_img1 = load('mosaic2_test.txt');
test_img2 = load('mosaic3_test.txt');

% Quantizing to G gray levels
G = 16; % grayscale levels
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(:)))));

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

```

```
[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};

% Training
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] = multivatiateGaussianTrainer(feats, ...
    train_mask2);
[labels3, means3, covs3] = multivatiateGaussianTrainer(feats, ...
    train_mask3);

% Classification
[class1] = multivatiateGaussianClassifier(feats, labels2, ...
    means2, covs2);
[class2] = multivatiateGaussianClassifier(feats, labels3, ...
    means3, covs3);

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

% Visualization
figure(2)
imagesc(class1); colorbar; title('Result on test image 1'); ...
    axis('square');
figure(3)
imagesc(class2); colorbar; title('Result on test image 2'); ...
    axis('square');
```