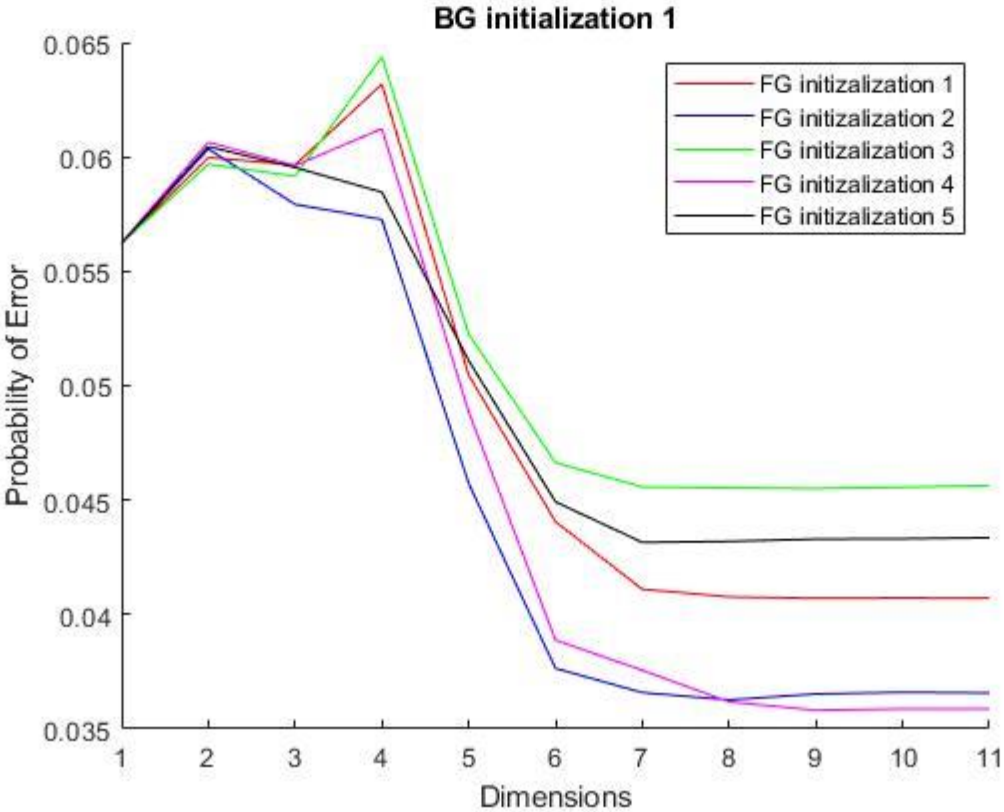
ECE 271A HW5
Computer Assignment Report
Arda Cankat Bati
Std id: A53284500

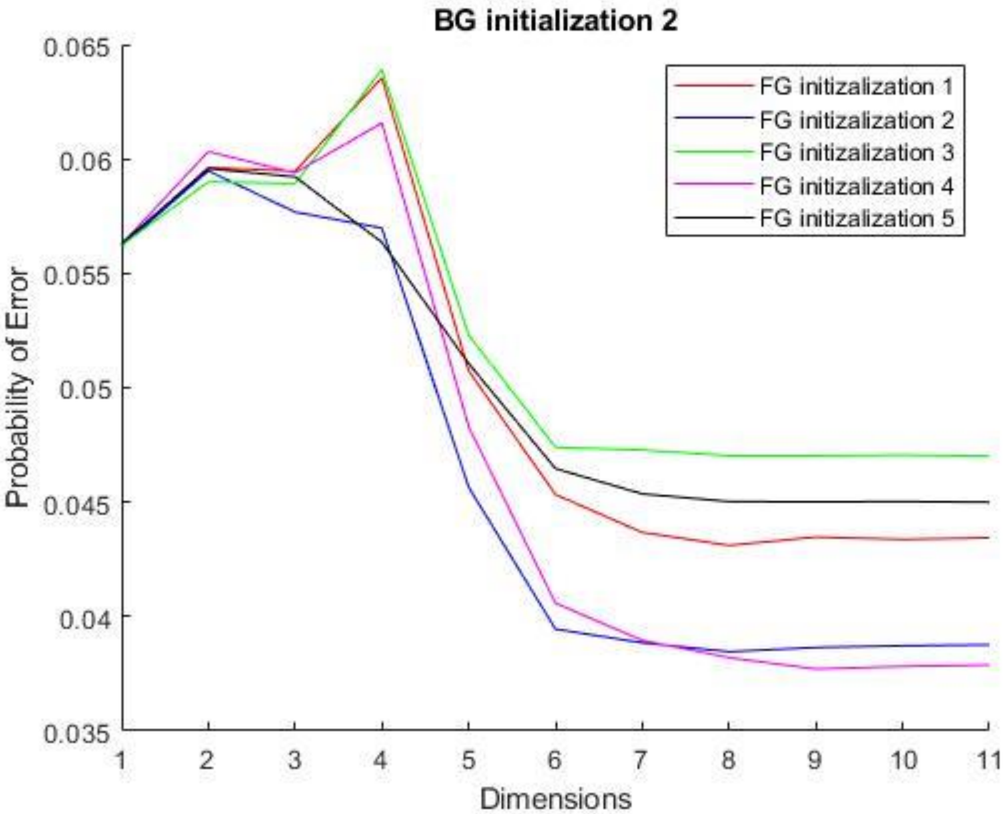
The MATLAB code written for HW is given at the end of the report. It consists of three parts, which should be used as separate .m files. The first & second ones are for Part 1 and Part 2 respectively. The last .m code contains the EM algorithm which is in function form. This function is called in the 1^{st} and 2^{nd} parts of the code.

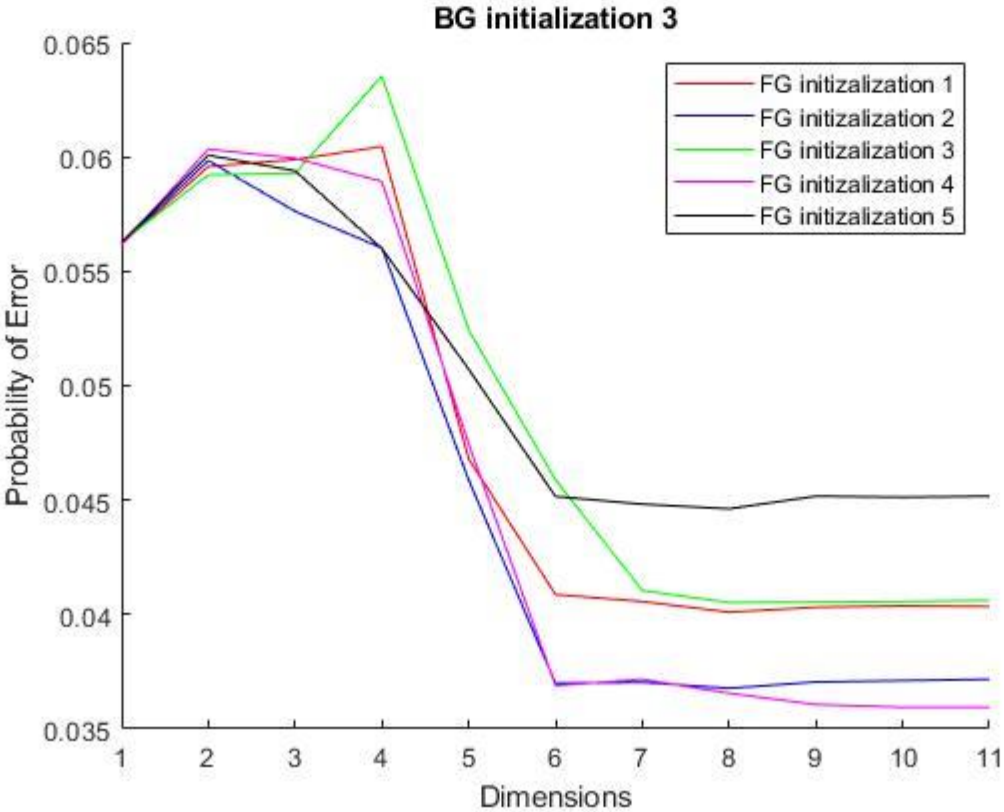
PART 1

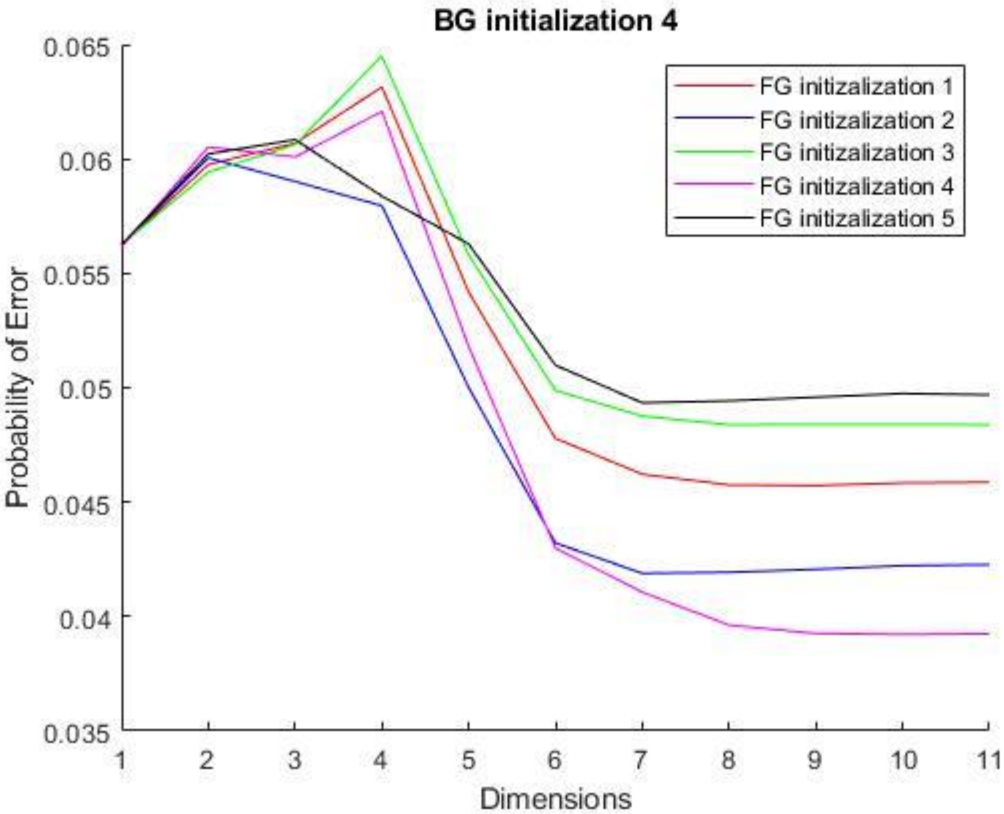
Below the graphs required in part 1 of the homework are given. There are 5 plots, for each initialization of the background class's mixtures. Here it is important to point out that the x-axis was modified for clarity. The numbers on the x-axis show the indices of the dimensions in the following vector: dim = [1, 2, 4, 8, 16, 24, 32, 40, 48, 56, 64]. For example, the number 5 on the x-axis corresponds to 16 dimensions being used.

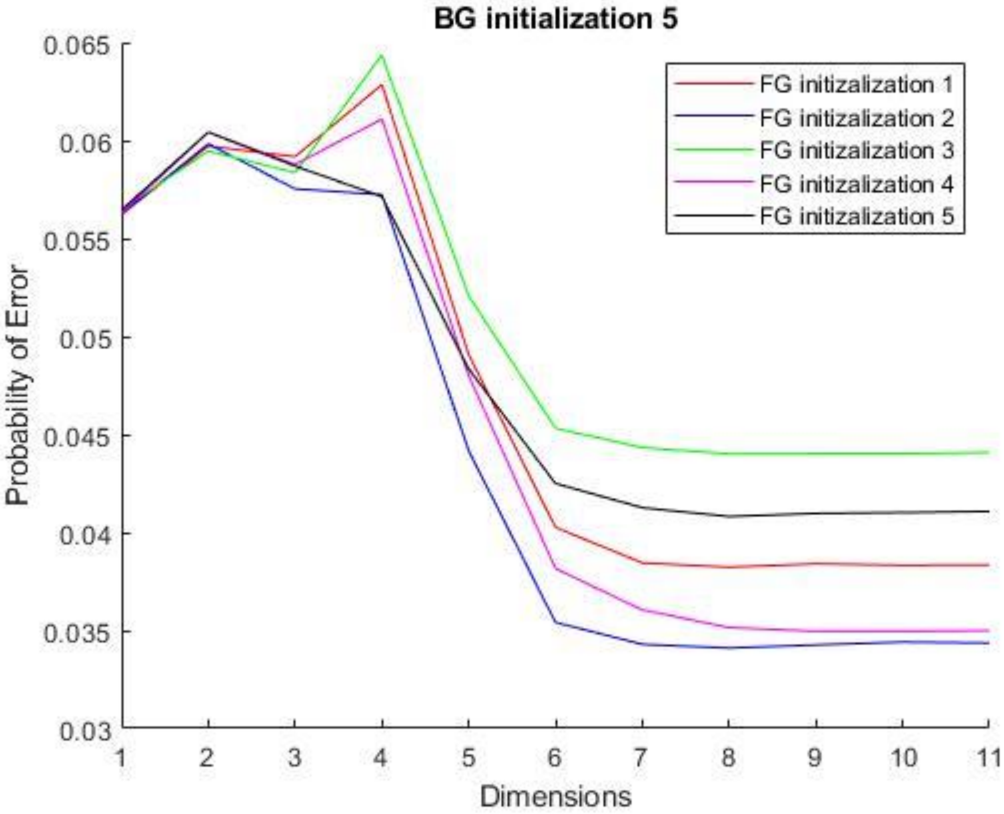
The initial means for the mixtures were calculated by k-means and they were used as input for the EM algorithm. As expected, in all the plots, and for all the lines, as the number of dimensions increase, the error decreases. We also observe that after a certain dimension number, namely dim[7] = 32 the error mostly stays flat. The overall tendency of the graph until this dimension usually determines the probability of error for the higher dimensions. We see that for different initializations, the graphs have mostly the same shape. For the dimensions [1, 2, 4] they usually show the same result. After more dimensions are included, the difference between the mixtures become more evident, and the plots start diverging. For dim[4] = 8, we usually see a peak in the graphs. This may have a relation with our component count = 8, however I can't find a clear reason for this behaviour. After dim[4] = 8, for the higher dimensions we see a raid fall in the error probabilities. We are including more info in the form of dimensions, and grouping them in the hidden groups. This way we are making more educated guesses about the image's underlying behaviour, therefore we are getting better results. The randomness of the starting parameters given to the EM algorithm causes the main difference between each initialization.


BG initialization 1
Probability of Error
Dimensions
FG initizalization 1
FG initizalization 2
FG initizalization 3
FG initizalization 4
FG initizalization 5
0.065
0.06
0.055
0.05
0.045
0.04
0.035
1
2
3
4
5
6
7
8
9
10
11


BG initialization 2
Probability of Error
Dimensions
FG initizalization 1
FG initizalization 2
FG initizalization 3
FG initizalization 4
FG initizalization 5
0.065
0.06
0.055
0.05
0.045
0.04
0.035
1
2
3
4
5
6
7
8
9
10
11


BG initialization 3
Probability of Error
Dimensions
FG initizalization 1
FG initizalization 2
FG initizalization 3
FG initizalization 4
FG initizalization 5
0.065
0.06
0.055
0.05
0.045
0.04
0.035
1
2
3
4
5
6
7
8
9
10
11


BG initialization 4
Probability of Error
Dimensions
FG initizalization 1
FG initizalization 2
FG initizalization 3
FG initizalization 4
FG initizalization 5
0.065
0.06
0.055
0.05
0.045
0.04
0.035
1
2
3
4
5
6
7
8
9
10
11

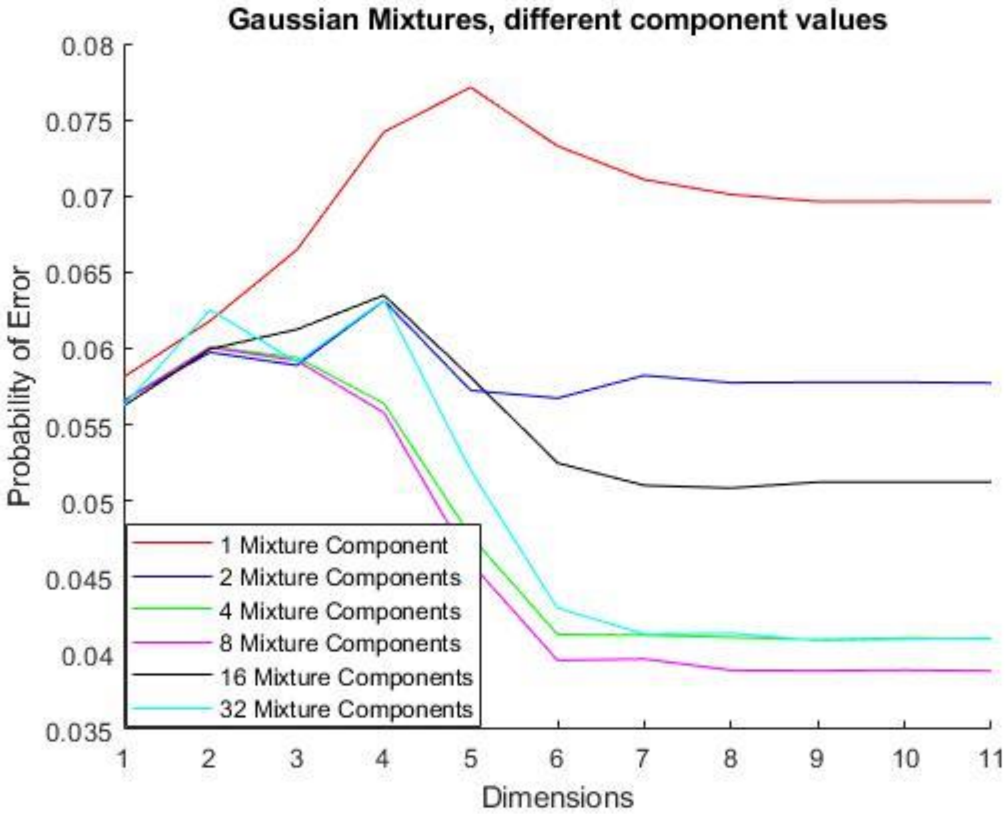


PART 2

The resulting error probability graph for this part is given below. Similar to the previous part, the x-axis values are the indices of the vector: dim = [1, 2, 4, 8, 16, 24, 32, 40, 48, 56, 64]. From our discussion in the lectures, we expect that as the component number increases, we should get better or similar results. While increasing the component number, after a certain value we reach the actual hidden class amount in the original distribution. After this point, even though we increase the component number further, our error should not be changing. The EM algorithm is very robust in this regard, converging to a valid solution, even when we have too many component classes. Also, similar to Part 1, we expect less error as we increase the number of dimensions. This is because we are including more information into our model. As we are considering every feature's effect for every hidden class in our EM algorithm, the more features we use, the better the model becomes. We don't have rigid limits as we had in ML, therefore with increasing number of dimensions, we learn the underlying distribution better.

In the plot below, we see the results are mostly in line with our expectations. As the component number increases, the probability of error for model tends to decrease. The only exception to this is the 16 component case. Also, the 8 component case has slightly less error than the 32 component case. These may be caused by the inherent randomness of our situation. More trials should be run on different train / test datasets to reach more concrete results.

Considering the dimension count, all the plots start at similar values for low dimension counts. Here as we have too few features, there is not much to assign to the hidden classes. For increasing number of dimensions, the error decreases, apart from 1 mixture and 2 mixture models. In these models, by only having 1 & 2 hidden classes, we are putting a too strict bound on our model. Therefore, as the number of dimensions increase, our error increases, or stays the same. Although we have more information coming from higher dimensions, we are not using it effectively. For 4 or more components, error always decreases with increasing number of dimensions. This is in line with our expectations.



```
% ****** HW5 Part 1 ******* %

clc
clear
load ('TrainingSamplesDCT_8_new.mat');

FG = TrainsampleDCT_FG;
BG = TrainsampleDCT_BG;
FG_size = size(FG,1);
BG_size = size(BG,1);
sampleSize = FG_size + BG_size;
CPrior = FG_size/(sampleSize);
NCPrior = BG_size/(sampleSize);

[cImageOld,colormap] = imread('cheetah.bmp');
cImage = im2double(cImageOld);
paddingType = 'replicate';
cImage = padarray(cImage,[4 4],paddingType,'pre');
cImage = padarray(cImage,[3 3],paddingType,'post');
[cImageReal colormap] = imread('cheetah_mask.bmp');

cImageReal = double(cImageReal)/255;
Image_Size = size(cImageReal,2)*size(cImageReal,1);
FG_Sum = sum(sum(cImageReal));
BG_Sum = Image_Size - FG_Sum;
cImageOldX = size(cImageOld,1); cImageOldY = size(cImageOld,2);
cImageX = size(cImage,1); cImageY = size(cImage,2);
A = [0   1   5   6   14   15   27   28 2   4   7   13   16   26   29   42 3   8   12
 17   25   30   41   43 9   11   18   24   31   40   44   53 10   19   23   32   39
 45   52   54 20   22   33   38   46   51   55   60 21   34   37   47   50   56   59
 61 35   36   48   49   57   58   62   63];
A = A + 1;

decisionImage64 = zeros(size(cImageOld,1),size(cImageOld,2));
points = zeros(cImageOldX*cImageOldY,64);
point = zeros(64,1);
count = 1;

for i = 1:cImageOldX
    for j = 1:cImageOldY
    temp = (dct2(cImage(i:i+7, j:j+7)))';
    vectorDct= temp(:);
    point(A) = vectorDct;
    points(count,:) = point';
    count = count + 1;
    end
end

dim = 64;
c = 8;

%For test purposes
```

```
p_size = 5;
%p_size = 1;

priors_FG = zeros(5,c);
means_FG = cell(5,1);
covs_FG = cell(5,1);

priors_BG = zeros(5,c);
means_BG = cell(5,1);
covs_BG = cell(5,1);


%FG class: Random initialization/kmeans and EM algorythm for 5
 different
%initializations
for num = 1:p_size
    num

    %Prior initialization
    start_prior = (ones(1,c));
    start_prior = start_prior / c;

    %Mean initialization by kmeans
    [labels, start_mean] = kmeans(FG, c);
    clear labels

    %Random covariance initialization
    start_cov = zeros(c,dim);

    cov_diag = rand(c,dim);
    cov_diag(cov_diag < 0.0005) = 0.0005;

    for component = 1 : c
        start_cov(component,:) = cov_diag(component, :);
    end

    %EM algorhtym implemented in seperate .m file
    [cur_mean, var, cur_prior] = EM(FG, c, start_mean, start_cov,
 start_prior);

    priors_FG(num, :) = cur_prior;
    means_FG{num} = cur_mean;
    covs_FG{num} = var;
end

%BG class: Random initialization/kmeans and EM algorythm for 5
 different
%initializations
for num = 1:p_size
    num

    %Prior initialization
    start_prior = (ones(1,c));
    start_prior = start_prior / c;
```

```
    %Mean initialization by kmeans
    [labels, start_mean] = kmeans(BG, c);
    clear labels

    start_cov = zeros(c,dim);

    %Random covariance initialization
    cov_diag = rand(c,dim);
    cov_diag(cov_diag < 0.0005) = 0.0005;

    for component = 1 : c
        start_cov(component,:) = cov_diag(component, :);
    end

    %EM algorhtym implemented in seperate .m file
    [cur_mean, var, cur_prior] = EM(BG, c, start_mean, start_cov,
 start_prior);

    priors_BG(num, :) = cur_prior;
    means_BG{num} = cur_mean;
    covs_BG{num} = var;
end

%This matrix will be used to store all error rates 11 x 5 x 5
error_matrix = zeros(11,5,5);

%For test purposes
dim = 64;
dim = 2;

dim_count = 0;

%Saving variables for repeated test cases
save('hw5_part1_variables.mat')

% Main loop to try decision function for each of the 25 mixtures, 11
% idmensions
for dim = [1, 2, 4, 8, 16, 24, 32, 40, 48, 56, 64]

        dim_count = dim_count + 1
        cur_cov_BG = cell(5);
        cur_mean_BG = cell(5);
        cur_cov_FG = cell(5);
        cur_mean_FG = cell(5);

        % Getting the relevant stored variables from the above part
 for
        % this iteration of the loop
        for mixture = 1:p_size
            m1 = covs_BG{mixture};
            m2 = means_BG{mixture};
            m3 = covs_FG{mixture};
            m4 = means_FG{mixture};
```

```
            cur_cov_BG{mixture} = m1(:, 1:dim);
            cur_mean_BG{mixture} = m2(:, 1:dim);
            cur_cov_FG{mixture} = m3(:, 1:dim);
            cur_mean_FG{mixture} = m4(:, 1:dim);
        end

        % Trying 5x5 each feature in this loop
        for mixFG = 1:5

            for mixBG = 1:5

                count = 1;
                decisionImage =
zeros(size(cImageOld,1),size(cImageOld,2));

                % Pixel class decision is done in the below loop
                for i = 1:cImageOldX

                    for j = 1:cImageOldY

                        point = points(count,1:dim);

                        %Calculations for the background probability

                        tot_prob_BG = 0;

                        cur_mean = cur_mean_BG{mixBG};
                        cur_sig = cur_cov_BG{mixBG};
                        cur_prior = priors_BG(mixBG,:);

                        %Summing each of the hidden class' likelihoods
                        for component = 1 : c
                            tot_prob_BG = tot_prob_BG + mvnpdf(point,
cur_mean(component, :), diag(cur_sig(component,:))) *
cur_prior(component);
                        end

                        %Calculations for the foreground probability

                        tot_prob_FG = 0;

                        cur_mean = cur_mean_FG{mixFG};
                        cur_sig = cur_cov_FG{mixFG};
                        cur_prior = priors_FG(mixFG,:);

                        %Summing each of the hidden class' likelihoods
                        for component = 1 : c
                            tot_prob_FG = tot_prob_FG + mvnpdf(point,
cur_mean(component, :), diag(cur_sig(component,:))) *
cur_prior(component);
                        end

                        %Main decision function
```

```
                       [M,decision] = max([(tot_prob_BG * NCPrior)
 (tot_prob_FG * CPrior)]);
                       decision = decision - 1;
                       decisionImage(i,j) = decision;
                       count = count + 1;
                   end

               end

               errorMaskBG = decisionImage - cImageReal;
               %FG misclassified as BG / total true FG
               beta_error = sum(sum(errorMaskBG == -1)) /
 FG_Sum; %False Negative --> beta
               %BG misclassified as FG / total true BG
               alpha_error = sum(sum(errorMaskBG == 1)) / BG_Sum;
  %False Positive --> alpha
               error_matrix(dim_count, mixBG, mixFG) = CPrior *
 beta_error + NCPrior * alpha_error;

%                 figure()
%                 I = mat2gray(decisionImage,[0 1]);
%                 imshow(I); title(sprintf('Prediction image'));

           end
       end
end

%Drawing the plots

for i = 1:5

    figure();
    x = [1:11];
    err1 = error_matrix(:, i, 1);
    err2 = error_matrix(:, i, 2);
    err3 = error_matrix(:, i, 3);
    err4 = error_matrix(:, i, 4);
    err5 = error_matrix(:, i, 5);

    hold on
    p1 = plot(x, err1, 'r'); L1 = "FG initialization 1";
    p2 = plot(x, err2, 'b'); L2 = "FG initialization 2";
    p3 = plot(x, err3, 'g'); L3 = "FG initialization 3";
    p4 = plot(x, err4, 'm'); L4 = "FG initialization 4";
    p5 = plot(x, err5, 'k'); L5 = "FG initialization 5";
    lgd = legend([p1,p2,p3,p4,p5], [L1, L2, L3, L4, L5]);
    lgd.Position = [0.75 0.8 0 0];
    title(sprintf('BG initialization %d',i));
    xlabel('Dimensions'); ylabel('Probability of Error');
    hold off

end
```

Published with MATLAB® R2018b

```
% ****** HW5 Part 2 ******* %

clc
clear
load ('TrainingSamplesDCT_8_new.mat');

FG = TrainsampleDCT_FG;
BG = TrainsampleDCT_BG;
FG_size = size(FG,1);
BG_size = size(BG,1);
sampleSize = FG_size + BG_size;
CPrior = FG_size/(sampleSize);
NCPrior = BG_size/(sampleSize);

[cImageOld,colormap] = imread('cheetah.bmp');
cImage = im2double(cImageOld);
paddingType = 'replicate';
cImage = padarray(cImage,[4 4],paddingType,'pre');
cImage = padarray(cImage,[3 3],paddingType,'post');
[cImageReal colormap] = imread('cheetah_mask.bmp');

figure();
imshow(cImage); title('Original image with symmetric padding.');


cImageReal = double(cImageReal)/255;
Image_Size = size(cImageReal,2)*size(cImageReal,1);
FG_Sum = sum(sum(cImageReal));
BG_Sum = Image_Size - FG_Sum;
cImageOldX = size(cImageOld,1); cImageOldY = size(cImageOld,2);
cImageX = size(cImage,1); cImageY = size(cImage,2);
A = [0   1   5   6   14   15   27   28 2   4   7   13   16   26   29   42 3   8   12
 17   25   30   41   43 9   11   18   24   31   40   44   53 10   19   23   32   39
 45   52   54 20   22   33   38   46   51   55   60 21   34   37   47   50   56   59
 61 35   36   48   49   57   58   62   63];
A = A + 1;

decisionImage64 = zeros(size(cImageOld,1),size(cImageOld,2));
points = zeros(cImageOldX*cImageOldY,64);
point = zeros(64,1);
count = 1;

for i = 1:cImageOldX
    for j = 1:cImageOldY
    temp = (dct2(cImage(i:i+7, j:j+7)))';
    vectorDct= temp(:);
    point(A) = vectorDct;
    points(count,:) = point';
    count = count + 1;
    end
end
```

```
%For test purposes
p_size = 6;
p_size = 2;

priors_FG = cell(6,1);
means_FG = cell(6,1);
covs_FG = cell(6,1);

priors_BG = cell(6,1);
means_BG = cell(6,1);
covs_BG = cell(6,1);

dim = 64;

component_sizes = [1, 2, 4, 8, 16, 32];

%FG class: Random initialization/kmeans and EM algorythm for 6
 different
%component sizes
for num = 1:p_size
    num
    c = component_sizes(num);

    %Prior initialization
    start_prior = (ones(1,c));
    start_prior = start_prior / c;

    %Mean initialization by kmeans
    [labels, start_mean] = kmeans(FG, c);
    clear labels

    %Random covariance initialization
    start_cov = zeros(c,dim);

    cov_diag = rand(c,dim);
    cov_diag(cov_diag < 0.0005) = 0.0005;

    for component = 1 : c
        start_cov(component,:) = cov_diag(component, :);
    end

    %EM algorhtym implemented in seperate .m file
    [cur_mean, var, cur_prior] = EM(FG, c, start_mean, start_cov,
 start_prior);

    priors_FG{num} = cur_prior;
    means_FG{num} = cur_mean;
    covs_FG{num} = var;
end

%BG class: Random initialization/kmeans and EM algorythm for 6
 different
%component sizes
for num = 1:p_size
```

```
    num
    c = component_sizes(num);

    %Prior initialization
    start_prior = (ones(1,c));
    start_prior = start_prior / c;

    %Mean initialization by kmeans
    [labels, start_mean] = kmeans(BG, c);
    clear labels

    %Random covariance initialization
    start_cov = zeros(c,dim);

    cov_diag = rand(c,dim);
    cov_diag(cov_diag < 0.0005) = 0.0005;

    for component = 1 : c
        start_cov(component,:) = cov_diag(component, :);
    end

    %EM algorhtym implemented in seperate .m file
    [cur_mean, var, cur_prior] = EM(BG, c, start_mean, start_cov,
 start_prior);

    priors_BG{num} = cur_prior;
    means_BG{num} = cur_mean;
    covs_BG{num} = var;
end

save('hw5_part2_variables.mat')

%This matrix will be used to store all error rates 11 x 5 x 5
error_matrix = zeros(6,11);

%Main loop to try 6 different component sizes for 11 dimensions


for component_count = 1:p_size

    % Getting the relevant stored variables from the above part for
    % this iteration of the loop

    c = component_sizes(component_count)
    m1 = covs_BG{component_count};
    m2 = means_BG{component_count};
    m3 = covs_FG{component_count};
    m4 = means_FG{component_count};

    dim_count = 0;
    for dim = [1, 2, 4, 8, 16, 24, 32, 40, 48, 56, 64]

            cur_cov_BG = m1(:, 1:dim);
            cur_mean_BG = m2(:, 1:dim);
```

```
            cur_cov_FG = m3(:, 1:dim);
            cur_mean_FG = m4(:, 1:dim);

            dim_count = dim_count + 1
%             cur_cov_BG = cell(5);
%             cur_mean_BG = cell(5);
%             cur_cov_FG = cell(5);
%             cur_mean_FG = cell(5);

            count = 1;
            decisionImage =
zeros(size(cImageOld,1),size(cImageOld,2));

            % Pixel class decision is done in the below loop
            for i = 1:cImageOldX

                for j = 1:cImageOldY

                    point = points(count,1:dim);

                    %Calculations for the background probability
                    tot_prob_BG = 0;

                    %Summing each of the hidden class' likelihoods
                    for component = 1 : c
                        tot_prob_BG = tot_prob_BG + mvnpdf(point,
cur_mean_BG(component, :), diag(cur_cov_BG(component,:))) *
priors_BG{component_count}(component);
                    end

                    %Calculations for the foreground probability

                    tot_prob_FG = 0;

                    %Summing each of the hidden class' likelihoods
                    for component = 1 : c
                        tot_prob_FG = tot_prob_FG + mvnpdf(point,
cur_mean_FG(component, :), diag(cur_cov_FG(component,:))) *
priors_FG{component_count}(component);
                    end

                    %Main decision function
                    [M,decision] = max([(tot_prob_BG * NCPrior)
(tot_prob_FG * CPrior)]);
                    decision = decision - 1;
                    decisionImage(i,j) = decision;
                    count = count + 1;
                end

            end

            errorMaskBG = decisionImage - cImageReal;
            %FG misclassified as BG / total true FG
```

```
            beta_error = sum(sum(errorMaskBG == -1)) / FG_Sum; %False
 Negative --> beta
            %BG misclassified as FG / total true BG
            alpha_error = sum(sum(errorMaskBG == 1)) / BG_Sum;  %False
 Positive --> alpha
            error_matrix(component_count, dim_count) = CPrior *
 beta_error + NCPrior * alpha_error;

%                 figure()
%                 I = mat2gray(decisionImage,[0 1]);
%                 imshow(I); title(sprintf('Prediction image'));

    end
end

save('hw5_part2_variables.mat')

figure();
x_dimension = [1:11];
err1 = error_matrix(1,:);
err2 = error_matrix(2,:);
err3 = error_matrix(3,:); %problem
err4 = error_matrix(4,:);
err5 = error_matrix(5,:); %problem
err6 = error_matrix(6,:); %problem

%component_sizes = [1, 2, 4, 8, 16, 32];
hold on
p1 = plot(x, err1, 'r'); L1 = "1 Mixture Component";
p2 = plot(x, err2, 'b'); L2 = "2 Mixture Components";
p3 = plot(x, err3, 'g'); L3 = "4 Mixture Components";
p4 = plot(x, err4, 'm'); L4 = "8 Mixture Components";
p5 = plot(x, err5, 'k'); L5 = "16 Mixture Components";
p6 = plot(x, err6, 'c'); L6 = "32 Mixture Components";
lgd = legend([p1,p2,p3,p4,p5,p6], [L1, L2, L3, L4, L5, L6]);
lgd.Position = [0.75 0.8 0.2 0.2];
title('Gaussian Mixtures, different component values');
xlabel('Dimensions'); ylabel('Probability of Error');
hold off
```

Published with MATLAB® R2018b

```
% ****** HW5 EM Algorithm ******* %

function [mean_result, variace_result, prior_result] = EM(dataset, c,
 mean, covariance, prior)
    [size_x size_y] = size(dataset);
    clear size_y

    % hij matrix from the slides
    hij = zeros(size_x, c);

    %z = diag(ones(c,1));

    prev_prior = prior; cur_prior = prior;
    prev_mean = mean; cur_mean = mean;
    var_prev = covariance; var_cur = covariance;

    for iterations = 1 : 100

        % E - STEP for Gaussian Mixtures, from the slides
        for row = 1 : size_x
            for component = 1:c
                hij(row, component) =  prev_prior(component)
 * mvnpdf(dataset(row, :), prev_mean(component, :),
 diag(var_prev(component, :)));
            end
            sum_hij = sum(hij(row, :));
            hij(row, :) = hij(row, :) / sum_hij;
        end

        % M -  STEP for Gaussian Mixtures, from the slides, Lagrangian
        % formulation was used in the slides to satisfy the constraint
 sum(pij)equal to 1
        for j = 1:c
            mj_raw = zeros(1,64);

            for i = 1:size_x
                mj_raw = mj_raw + hij(i,j) * dataset(i,:);
            end

            %Calculating the next priors and mean
            row_sum = sum(hij(:, j));

            %New means
            cur_mean(j, :) = mj_raw./row_sum;
            %New priors
            cur_prior(j) = row_sum / size_x;

        end

        % This part is seperated from above for easier debugging
        % Should be merged with the above part
```

```
        for j = 1:c

            sigmaj_raw = 0;

            for i=1: size(dataset,1)
                square_diff = (dataset(i,:)-cur_mean(j,:)).^2;
                sigmaj_raw = sigmaj_raw + hij(i,j) * square_diff;
            end
            row_sum = sum(hij(:, j));
            %New covariance below
            var_cur(j, :) = sigmaj_raw / row_sum;
            %Regularizing the covariance to prevent possible problems
 from 0 values
            var_cur(var_cur < 0.0005) = 0.0005;
        end
        %For test purposes
%         value = abs(cur_mean - prev_mean);
%         condition = mean(mean(value));
        prev_mean = cur_mean;
        prev_prior = cur_prior;
        var_prev = var_cur;

    end

    prior_result = prev_prior;
    mean_result = prev_mean;
    variace_result = var_prev;
```

Published with MATLAB® R2018b