a)

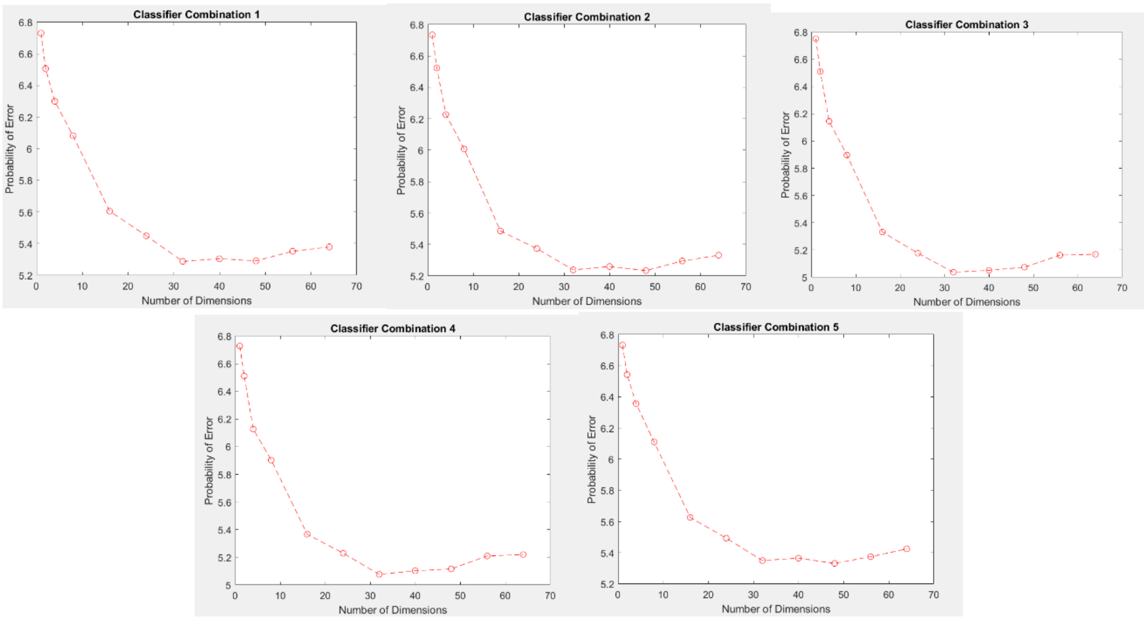

Figure 1: Probability of error vs number of dimensions for classifier combinations 1 - 5

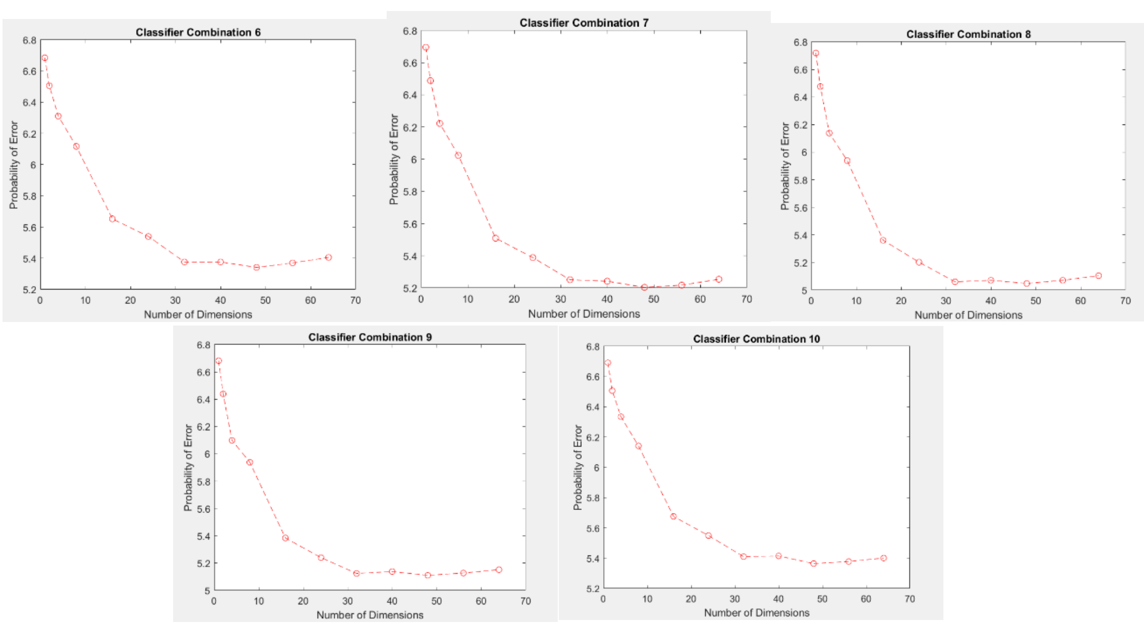

Figure 2: Probability of error vs number of dimensions for classifier combinations 6 - 10

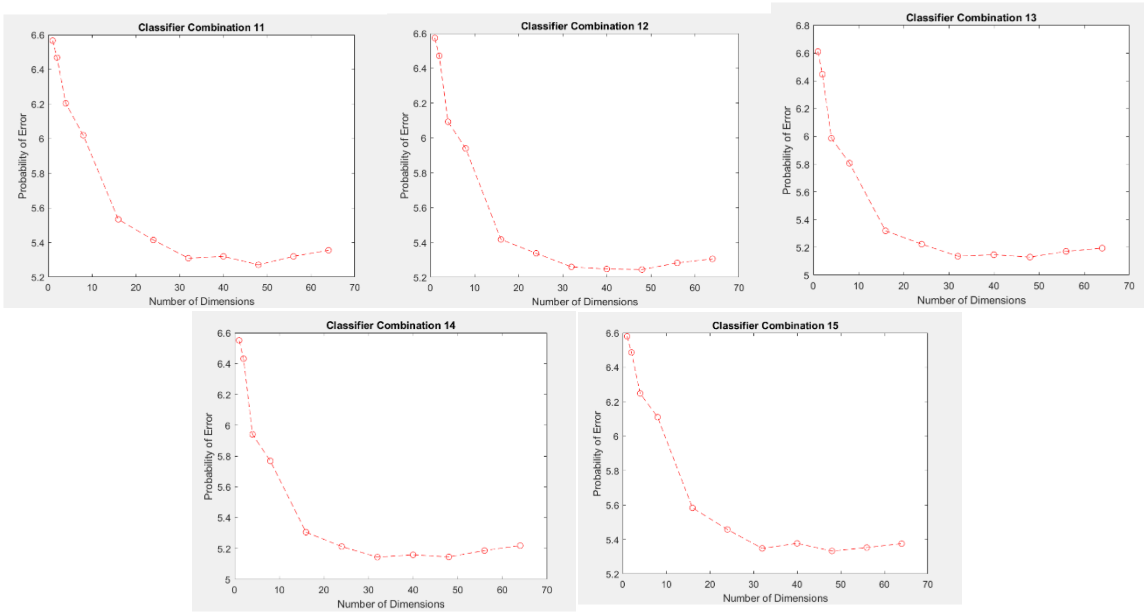

Figure 3: Probability of error vs number of dimensions for classifier combinations 11 - 15

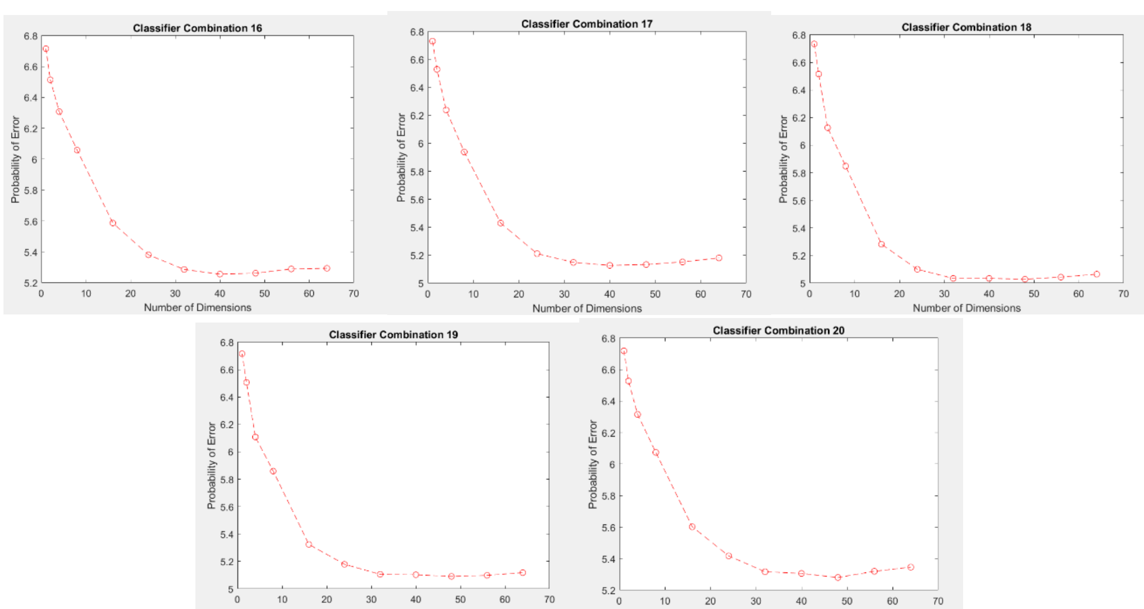

Figure 4: Probability of error vs number of dimensions for classifier combinations 16 - 20

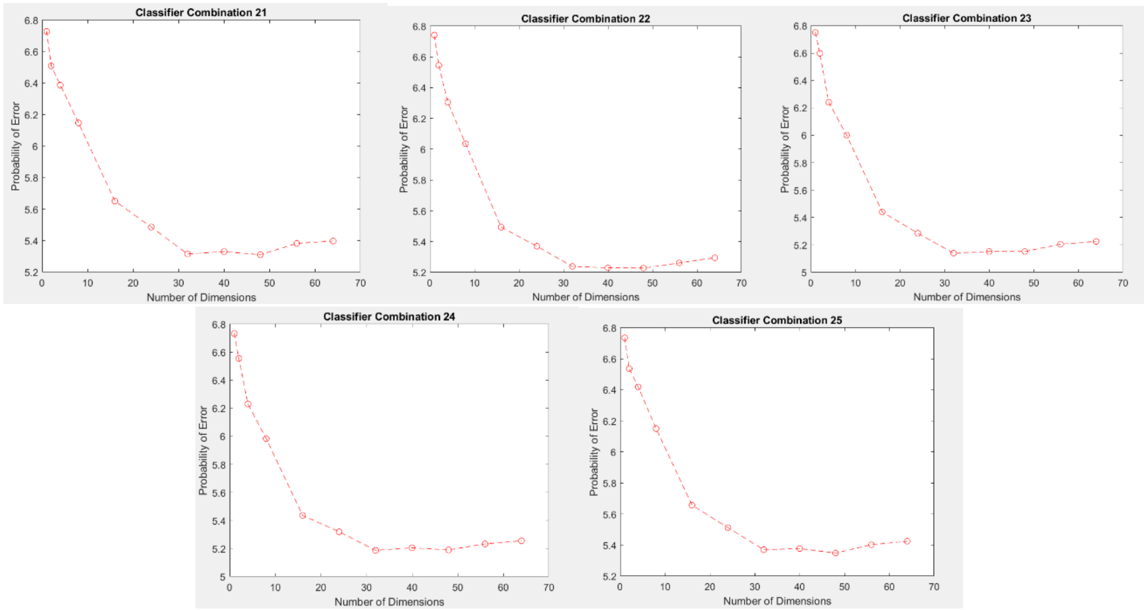

Figure 5: Probability of error vs number of dimensions for classifier combinations 21 – 25

All 25 combinations of classifiers generally follow the same trend: as the number of dimensions increases the error percentage decreases, but after about 30 (in most cases) or 40 (in a few cases) dimensions are used the error percentage begins to increase. There are only 2 combinations that result in a continuous decrease in error percentage up to 48 dimensions. This behavior is likely due to overfitting. Based on the extra computation time it took to run the classifications for >30 dimensions, and the overall benefit of using more dimensions, it is not worth performing classifications past 30 dimensions as the error percentage only decreased by a miniscule amount in a few combinations and increased in the vast majority of combinations. The minimum values of error percentage ranged from 5.0281% (combination 18) to 5.3641% (combination 10) and the maximum values of error percentage ranged from 6.5509% (in combination 14) to 6.7510% (combination 3).

The only aspect that was different between the classification combinations was the random initialization of the classifier parameters. It is clear that the initialization step for the parameters is important as the approximately 0.3% and 0.2% difference in the minimum and maximum error percentages respectively is not negligible in most classification/machine learning applications. I created a distribution of values and randomly selected values from the distribution to initialize the parameters. I provided plots of the cheetah classifications for combination 1 in Figure 6, and as one can see the random initialization works quite well. However, using k-means would be a more robust approach to parameter initialization and would aid in minimizing the error percentage difference between combinations.

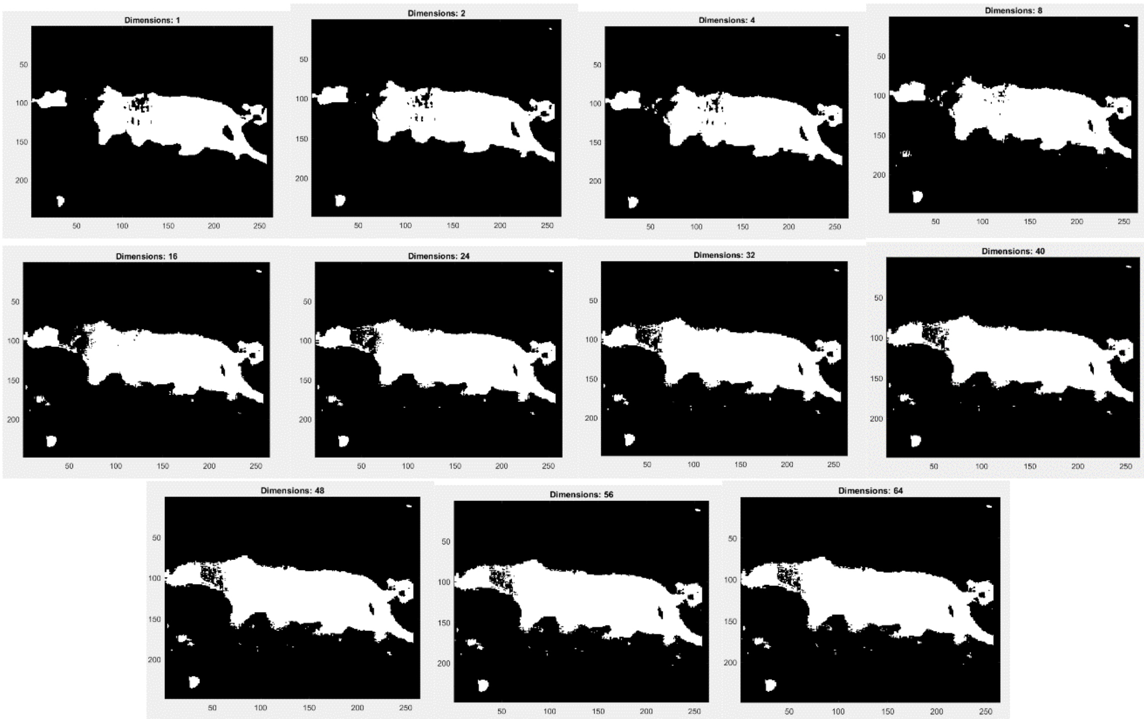

Figure 6: Classification results for varying dimensions of combination 1

As a side note, I used the same number of iterations (300) for learning parameters. This number was chosen by running approximately 20 convergence tests to observe how many iterations were generally required for convergence. The number of iterations ranged from approximately 75 to 250 in the tests I ran – so just to be safe I set the number of iterations to 300 to ensure everything converged. A delta limit would be a more robust approach, however for the sake of this assignment setting a high iteration count worked just fine and the extra computation time was negligible. Figure 7 displays an example of one of the convergence tests.

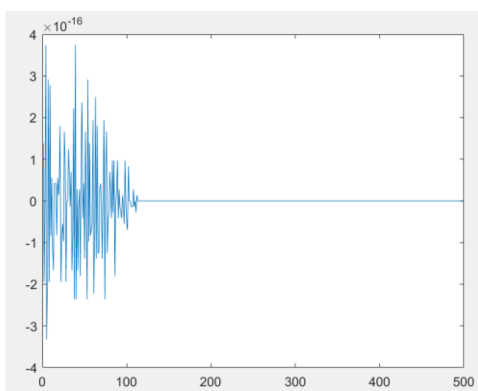

Figure 7: Convergence test example (delta of parameters vs iteration)

b)

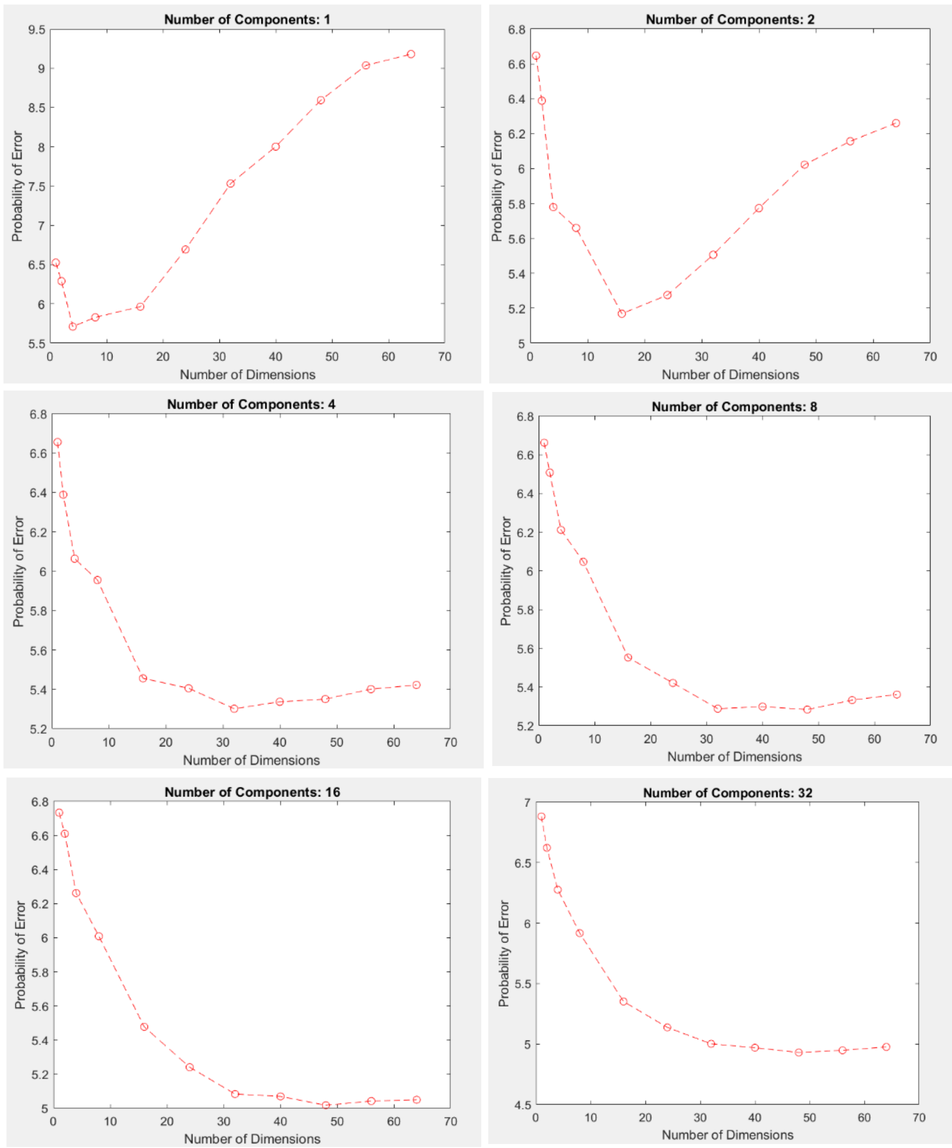

Figure 8: Probability of error vs number of dimensions for varying components

As one can see in Figure 8, the number of dimensions has varying affects on the error percentage as the number of components used in the gaussian mixture model varies. When the number of components is minimal (1 and 2) the error percentage initially decreases as extra dimensions are used, but then begins to increase once 8 and 24 dimensions are used for 1 and 2 components respectively. The increase in the error percentage is most dramatic when only 1 component is used as the error percentage reaches a value of approximately 9.3% at 64 dimensions as opposed to approximately 6.3% when 2 components are used. Although the error percentage increases after 16 dimensions in the 2 component mixture model, the error percentage at 64 dimensions is still smaller than the error percentage at the initial 1 dimensions by approximately 0.4%. Whereas in the 1 component mixture model, the error percentage at 64 dimensions is approximately 2.75% greater than the initial 1 dimension error percentage.

As the number of components increases to 4 and 8, the additional dimensions further decrease the error percentage up to about 30 dimensions. When more than 30 dimensions are used for mixture models containing 4 and 8 components the error percentage begins to increase. Once 16 and 32 components are used, however, the extra dimensions decrease the error percentage up until 48 dimensions are used. When more than 48 dimensions are used, once again, the error percentage begins to increase. This is likely due to overfitting the data. The best results are observed when 32 components are used as the minimum error percentage is approximately 4.9%. The minimum error percentage continues to get smaller (i.e. the classification is better) as components are added, but the decrease in error percentage is especially small between the increase of 16 to 32 components and the extra computation time required for 32 components as opposed to 16 components was not negligible.

Code

```
%Joseph Bell
%ECE271 HW5

clc;
clear;
load('TrainingSamplesDCT_8_new.mat');


num_mixtures = 5;
num_components = 8;
%8 components part


% start of loop to learn parameters for each component for FG

%Using containers for easy way to store values and access them later
learned_mu_FG = containers.Map; %should hold num_mixtures (5) items
learned_covariance_FG = containers.Map;
learned_weights_FG = containers.Map;

for mix=1:num_mixtures
    mu_c_FG = [];
    covariance_c_FG = []; %storing covariance as 1d array diag it later
    weights = [];
    disp(['Randomly initializing cheetah parameters for mixture: ' num2str(mix)]);
    for i=1:num_components
         %initialized random covariance and mu
        mu_c_FG = [mu_c_FG; normrnd(2,0.2,[1,64])]; %random mu
        covariance_c_FG = [covariance_c_FG; abs(normrnd(3,0.1,[1,64]))]; %random
covariance
        weights = [weights, 1/num_components]; %giving all initial equal weights
    end
        %gaussian function - taken from my last homework

    %tried this function I made but dimensions were not correct so used mvnpdf
    %and it worked
    %fun_cheetah = @(x, i) 1/sqrt((det(diag(covariance_c_FG(i,:)))*(2*pi)^64))*exp(-
1/2*(x-mu_c_FG(i,:))*inv(diag(covariance_c_FG(i,:)))*transpose(x-mu_c_FG(i,:)));

    %calculating weights for all 8 components for num of iterations
    sum_diff_of_weights = [];

    %ran a bunch of samples and found most converge by about 100-200, but
    %some took more so just in case made it 300 iterations. The amount of extra
    %time for the extra iterations is negligible

    disp(['Learning cheetah parameters for mixture: ' num2str(mix)]);
    num_iterations = 300;
    for iter=1:num_iterations
        prob_x_given_c_times_weight = [];
        for i=1:num_components
            result = mvnpdf(TrainsampleDCT_FG, mu_c_FG(i,:),
diag(covariance_c_FG(i,:))); %returns 250 by 1
            result_times_weight = result.*weights(1,i); %multiply each prob by weight
for component
            prob_x_given_c_times_weight = [prob_x_given_c_times_weight
result_times_weight];
        end
```

```
        %after this loop I have a 250x8 need to convert to a 1x8 via summation and
        %normalization to ensure the sum of the weights is 1

        %divide each column by sum of rows
        sum_rows = sum(prob_x_given_c_times_weight,2);
        prob_c_given_x = prob_x_given_c_times_weight./sum_rows;

        %above line gives me P(c|x) in 250*8 form
        sum_columns = sum(prob_c_given_x,1);
        new_weights = sum_columns/250;
        sum_diff_of_weights = [sum_diff_of_weights sum(new_weights-weights)];
        weights = new_weights;
        check_add_to_one = sum(new_weights);

        %modifying mean
        mu_c_FG = [];

        for i=1:num_components
             new_mu =
sum(prob_c_given_x(:,i).*TrainsampleDCT_FG)./sum(prob_c_given_x(:,i));
             mu_c_FG = [mu_c_FG; new_mu];

        end

        covariance_c_FG = [];
        %modifying covariance using new mean
        for i=1:num_components
            new_covariance = sum(prob_c_given_x(:,i).*(TrainsampleDCT_FG -
mu_c_FG(i,:)).^2);
            new_covariance = new_covariance./sum(prob_c_given_x(:,i));
            covariance_c_FG = [covariance_c_FG; abs(new_covariance)];
        end

    end
    %figure(mix)
    %plot(linspace(1,num_iterations,num_iterations),sum_diff_of_weights);
    learned_mu_FG(num2str(mix)) = mu_c_FG;
    learned_covariance_FG(num2str(mix)) = covariance_c_FG;
    learned_weights_FG(num2str(mix)) = weights;
    % learned_covariance_FG stores covariance as 1x64, must diag()
    % to use in pdf
end




%Using containers for easy way to store values and access them later
learned_mu_BG = containers.Map; %should hold num_mixtures (5) items
learned_covariance_BG = containers.Map;
learned_weights_BG = containers.Map;

% start of loop to learn parameters for each component for BG
for mix=1:num_mixtures
    mu_c_BG = [];
    covariance_c_BG = []; %storing covariance as 1d array diag it later
    weights = [];
    disp(['Randomly initializing grass parameters for mixture: ' num2str(mix)]);
    for i=1:num_components
        %initialized random covariance and mu
        mu_c_BG = [mu_c_BG; normrnd(3,0.3,[1,64])]; %random mu
```

```
        covariance_c_BG = [covariance_c_BG; abs(normrnd(3,0.1,[1,64]))]; %random
covariance
        weights = [weights, 1/num_components]; %giving all initial equal weights
    end
        %gaussian function - taken from my last homework

    %tried this function I made but dimensions were not correct so used mvnpdf
    %and it worked
    %fun_cheetah = @(x, i) 1/sqrt((det(diag(covariance_c_BG(i,:)))*(2*pi)^64))*exp(-
1/2*(x-mu_c_BG(i,:))*inv(diag(covariance_c_BG(i,:)))*transpose(x-mu_c_BG(i,:)));

    %calculating weights for all 8 components for num of iterations
    sum_diff_of_weights = [];

    %ran a bunch of samples and found most converge by about 100-200, but
    %some took more so just in case made it 300 iterations. The amount of extra
    %time for the extra iterations is negligible

    num_iterations = 300;
    disp(['Learning grass parameters for mixture: ' num2str(mix)]);
    for iter=1:num_iterations
        prob_x_given_c_times_weight = [];
        for i=1:num_components
            result = mvnpdf(TrainsampleDCT_BG, mu_c_BG(i,:),
diag(covariance_c_BG(i,:))); %returns 250 by 1
            result_times_weight = result.*weights(1,i); %multiply each prob by weight
for component
            prob_x_given_c_times_weight = [prob_x_given_c_times_weight
result_times_weight];
        end
        %after this loop I have a 250x8 need to convert to a 1x8 via summation and
        %normalization to ensure the sum of the weights is 1

        %divide each column by sum of rows
        sum_rows = sum(prob_x_given_c_times_weight,2);
        prob_c_given_x = prob_x_given_c_times_weight./sum_rows;

        %above line gives me P(c|x) in 250*8 form
        sum_columns = sum(prob_c_given_x,1);
        new_weights = sum_columns/250;
        sum_diff_of_weights = [sum_diff_of_weights sum(new_weights-weights)];
        weights = new_weights;
        check_add_to_one = sum(new_weights);

        %modifying mean
        mu_c_BG = [];

        for i=1:num_components
             new_mu =
sum(prob_c_given_x(:,i).*TrainsampleDCT_BG)./sum(prob_c_given_x(:,i));
             mu_c_BG = [mu_c_BG; new_mu];

        end

        covariance_c_BG = [];
        %modifying covariance using new mean
        for i=1:num_components
            new_covariance = sum(prob_c_given_x(:,i).*(TrainsampleDCT_BG -
mu_c_BG(i,:)).^2);
            new_covariance = new_covariance./sum(prob_c_given_x(:,i));
            covariance_c_BG = [covariance_c_BG; abs(new_covariance)];
        end
```

```
    end
    %figure(mix)
    %plot(linspace(1,num_iterations,num_iterations),sum_diff_of_weights);
    learned_mu_BG(num2str(mix)) = mu_c_BG;
    learned_covariance_BG(num2str(mix)) = covariance_c_BG;
    learned_weights_BG(num2str(mix)) = weights;
    % learned_covariance_BG stores covariance as 1x64, must diag()
    % to use in pdf
end

[row_FG, col_FG] = size(TrainsampleDCT_FG);
[row_BG, col_BG] = size(TrainsampleDCT_BG);

prior_FG = row_FG/(row_FG+row_BG);
prior_BG = row_BG/(row_FG+row_BG);

cheetah_mask = imread('cheetah_mask.bmp');
cheetah_mask = im2double(cheetah_mask);
cheetah_img = imread('cheetah.bmp');
cheetah_img = im2double(cheetah_img); %converting to double values since training data
is of type double
[cheetah_rows, cheetah_cols] = size(cheetah_img);
cheetah_img = cheetah_img(1:8*floor(cheetah_rows/8),1:8*floor(cheetah_cols/8));
%modifying image so it can be split into 8x8 blocks
cheetah_mask = cheetah_mask(1:8*floor(cheetah_rows/8),1:8*floor(cheetah_cols/8));
[cheetah_rows, cheetah_cols] = size(cheetah_img); %overwriting for modified dimensions


dimensions = [1 2 4 8 16 24 32 40 48 56 64];
[row_dim, col_dim] = size(dimensions);
counter = 0;
error_storage = containers.Map;
for mix_FG=1:num_mixtures
    mix_index_FG = num2str(mix_FG);
    cov_FG = learned_covariance_FG(mix_index_FG);
    mu_FG = learned_mu_FG(mix_index_FG);
    weights_FG = learned_weights_FG(mix_index_FG);
    for mix_BG=1:num_mixtures
        counter = counter + 1;
        disp('Beginning Classification Process');
        mix_index_BG = num2str(mix_BG);
        cov_BG = learned_covariance_BG(mix_index_BG);
        mu_BG = learned_mu_BG(mix_index_BG);
        weights_BG = learned_weights_BG(mix_index_BG);
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        new_image = zeros(cheetah_rows, cheetah_cols);
        pct_error = [];
        for dim=1:col_dim
            num_dimensions = dimensions(dim);
            disp(['Beginning dimension ' num2str(num_dimensions) ' classification']);
            for i=1:cheetah_cols-7 %shift scan pointer over a column
                for j=1:cheetah_rows-7
                    block = cheetah_img(j:7+j,i:7+i); %grab 8x8 block
                    block_dct = dct2(block);
                    zzblock_dct = zigzag(block_dct);
                    zzblock_dct = zzblock_dct(1,1:num_dimensions);
                    cheetah_result = 0;
                    grass_result = 0;
                    for k=1:num_components
                        cheetah_pd =
mvnpdf(zzblock_dct,mu_FG(k,1:num_dimensions),diag(cov_FG(k,1:num_dimensions)));
                        cheetah_result = cheetah_result + cheetah_pd*weights_FG(1,k);
```

```
                        grass_pd =
mvnpdf(zzblock_dct,mu_BG(k,1:num_dimensions),diag(cov_BG(k,1:num_dimensions)));
                        grass_result = grass_result + grass_pd*weights_BG(1,k);
                    end

                    choose_cheetah = cheetah_result*prior_FG;
                    choose_grass = grass_result*prior_BG;

                    if choose_cheetah > choose_grass
                        new_image(j:j,i:i) = 1;
                    end

                end
            end

            counter_correct = 0;
            total_pixels = cheetah_rows*cheetah_cols;
            for i=1:cheetah_rows
                for j=1:cheetah_cols
                    if cheetah_mask(i,j) ==  new_image(i,j)
                        counter_correct = counter_correct + 1;
                    end
                end
            end

            percent_correct = counter_correct/total_pixels*100;
            percent_error = 100 - percent_correct;
            pct_error = [pct_error percent_error];

            %{
            if counter == 1
                figure()
                imagesc(new_image)
                colormap(gray(255))

                title(['Dimensions: ' num2str(num_dimensions)]);
            end
            %}
        end
        disp(counter);
        error_storage(num2str(counter))= pct_error;
    end
end

for i=1:counter
    prob_error = error_storage(num2str(i));
    figure()
    plot(dimensions, prob_error, 'r--o');
    xlabel('Number of Dimensions');
    ylabel('Probability of Error');
    title(['Classifier Combination ' num2str(i)]);
end
```