Parthasarathi Kumar
A59003519

Homework 5

ECE 271A
December 2, 2021

Problem 6

(a) In this part, we train 5 Gaussian mixture models (GMMs) for each class. Each GMM has 8 components and has an independent random initialization. Figure 1-5 are the 5 plots for each GMM for foreground class against the 5 GMMs for the background class.

We see similar trends for each pair where probability of error decreases as the dimensions of the feature space increases. However, some GMMs have higher probability of errors. A possible reason is the random initialization causes it to converge to a local minima that is not the best set of parameters. As is obvious from the plots, there does exist set of parameters where the performance is better.

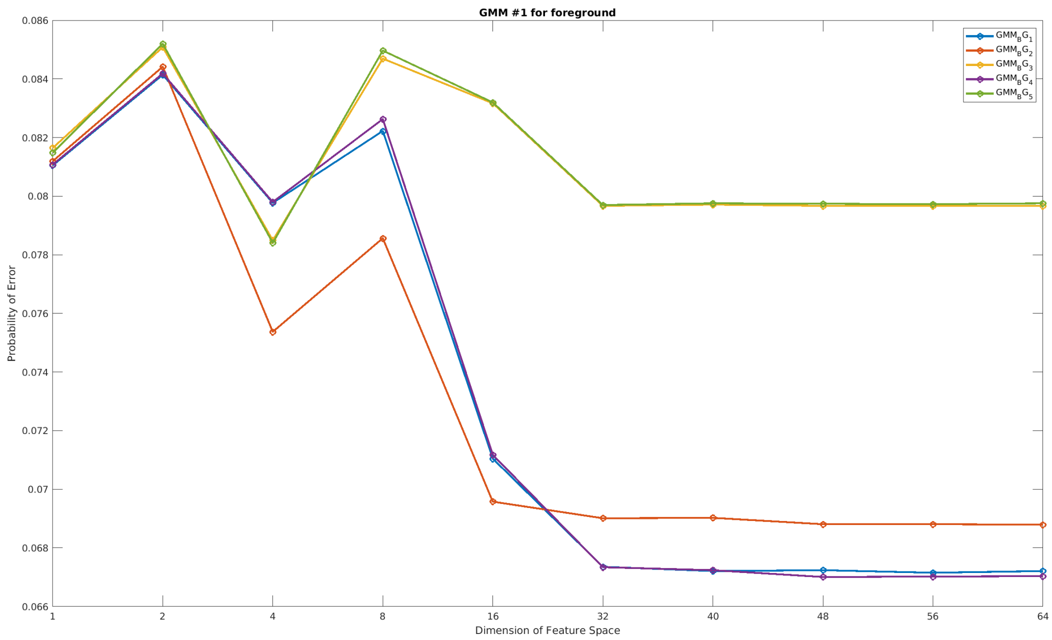

Figure 1: P(Error) vs Dimensions for GMM_1 for foreground

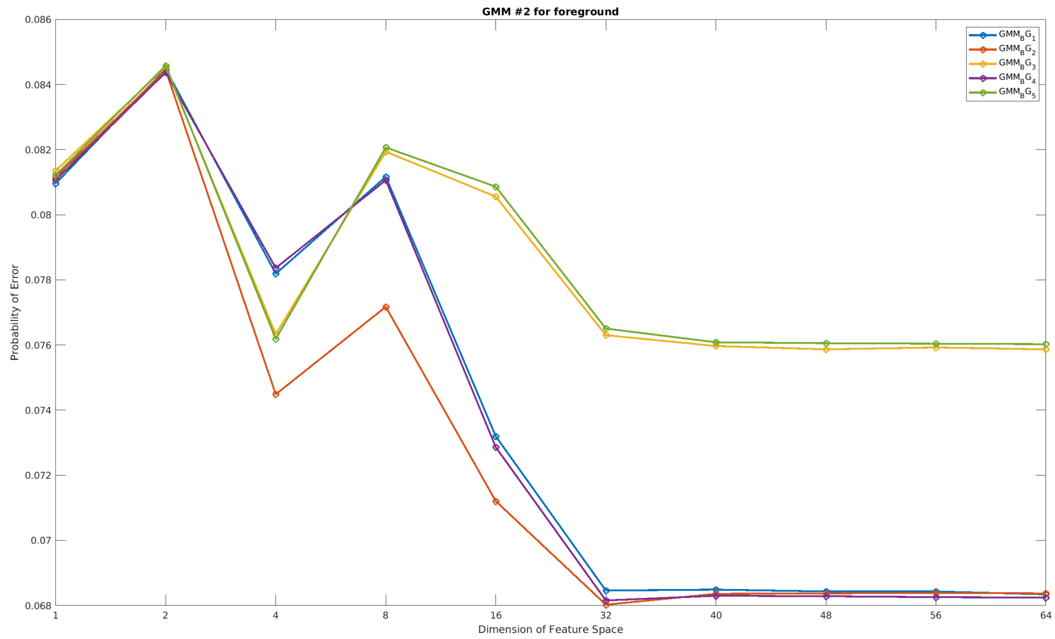

Figure 2: P(Error) vs Dimensions for GMM_2 for foreground

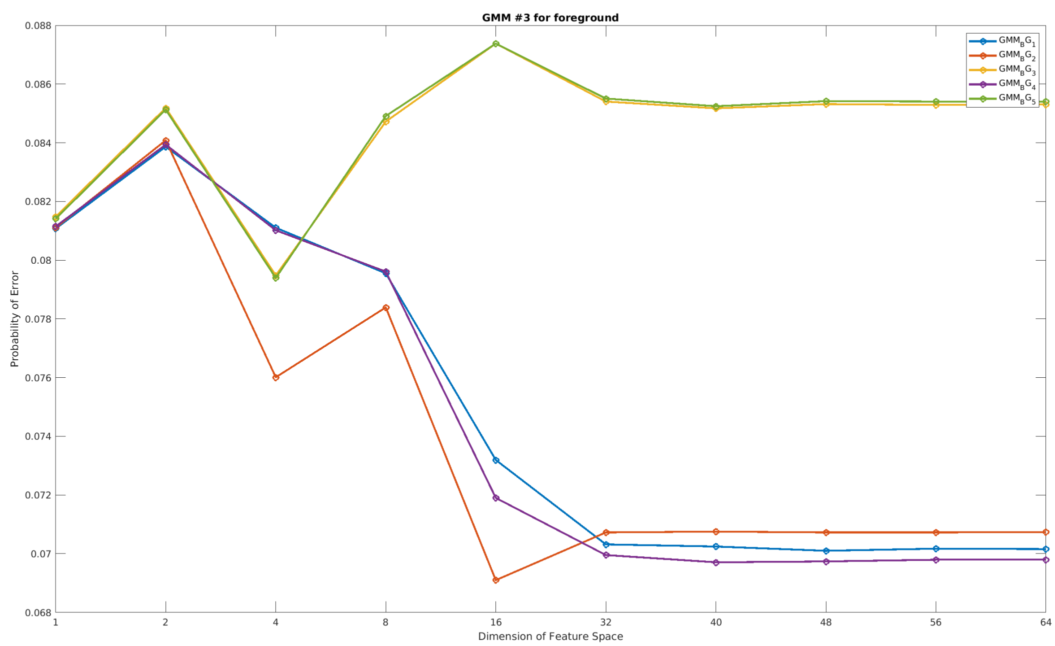

Figure 3: P(Error) vs Dimensions for GMM_3 for foreground

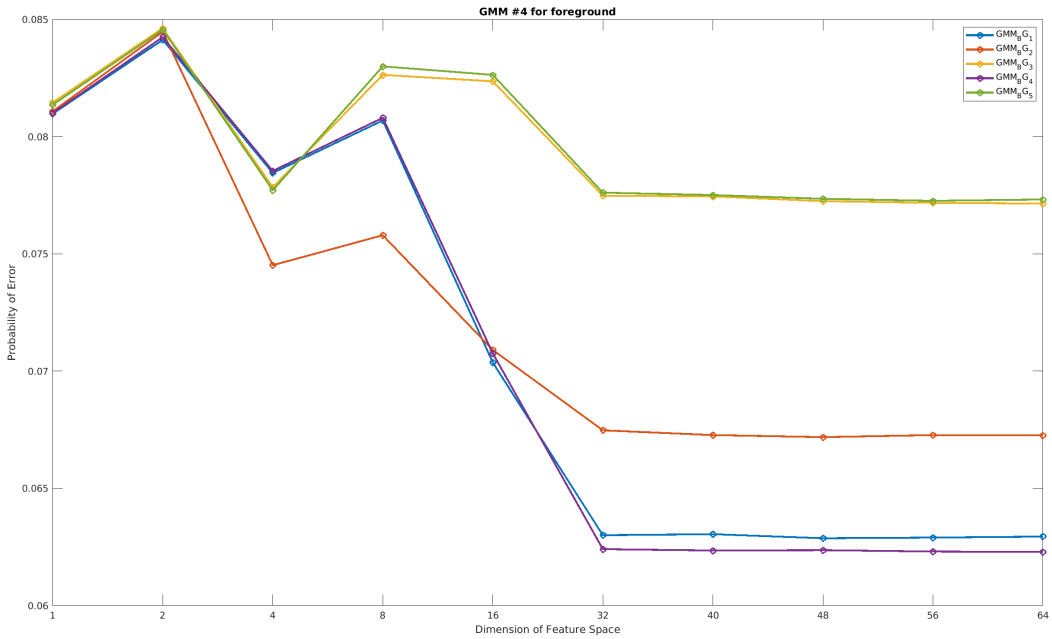

Figure 4: P(Error) vs Dimensions for GMM_4 for foreground

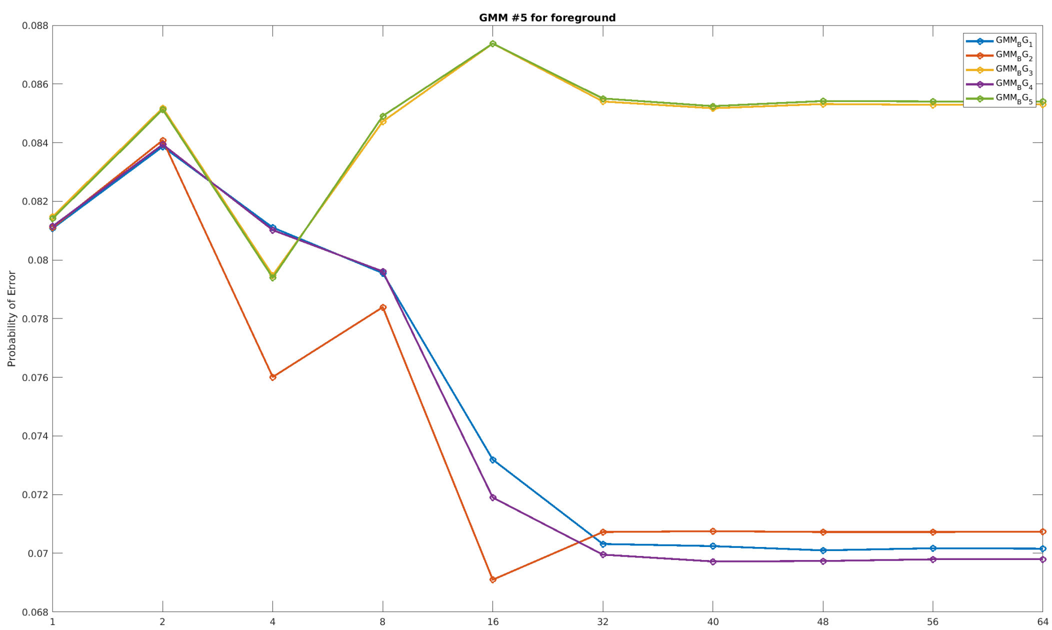

Figure 5: P(Error) vs Dimensions for GMM_5 for foreground

(b) In this part, we train 5 sets of Gaussian mixture models with the number of components C $\in$ {2,4,8,16,32}. Each pair is used on the test cheetah image with varying number of feature space dimension. Following is the plot of probability of error vs number of dimensions.

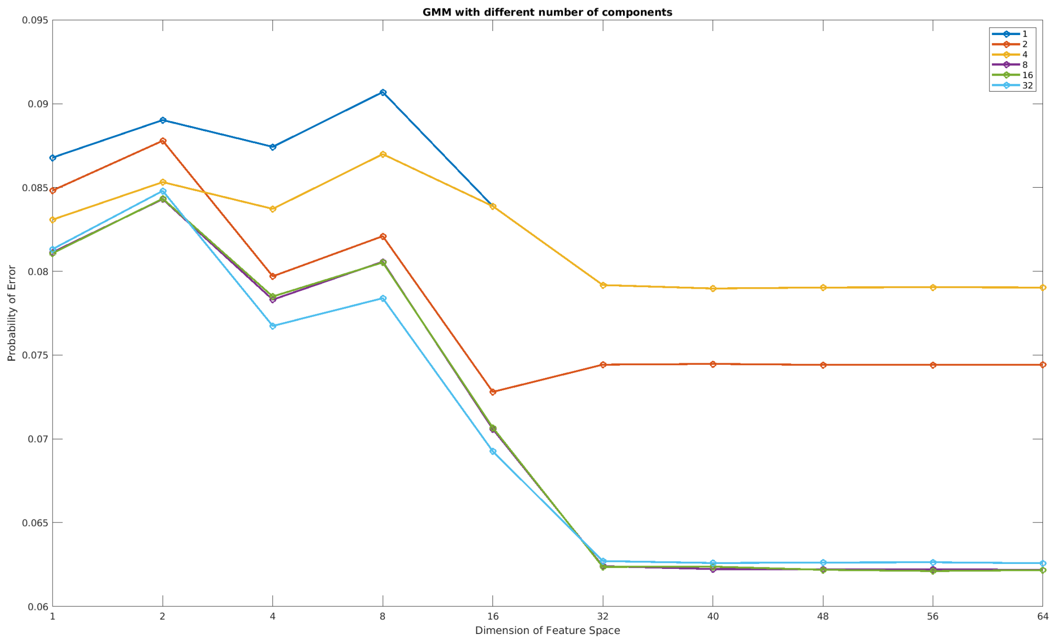

Figure 6: P(Error) vs Dimensions for GMM with different components

We observe that the probability of error reduces with more number of components. With C $\geqslant$8, the probability of errors for a given dimension of feature space is almost equal. This probability of error reduces as we increase the number of feature dimensions and saturates after a certain point(D $\geqslant$ 32).

MATLAB Code - Gaussian Mixture Model

```
classdef GaussianMixtureModel < handle

    % GaussianMixtureModel - class to train a Gaussian mixture model and
    % use it to predict the class conditional probability.
    % gmm = GaussianMixtureModel(C,D,threshold, maxIters, verbose) to
    % create a GMM object.
    % gmm.train(data) - To train GMM with data
    % gmm.predict(x) - To calculate probability.

    properties
        % Parameters to define a GMM
        Components
        Dimension
        Threshold
        Verbose
        MaxIters
    end

    properties
        % Parameters to store the trained parameters values
        Params
    end

    methods
        % ---------------------------------------------------------------------------

        % Constructor
        function this = GaussianMixtureModel(C, D, threshold, maxIters, verbose)
            this.Components = C;
            this.Dimension = D;
            this.Threshold = threshold;
            this.MaxIters = maxIters;
            this.Verbose = verbose;
        end

        % ---------------------------------------------------------------------------

        function train(this, data)
            %% Parameters for training GMM
            % Mixture components
            C = this.Components;
```

```
            % Dimension of feature space
            D = this.Dimension;
            % Threshold for converging
            threshold = this.Threshold;

            %% Gaussian Mixture Model

            % Taking first D features.
            data = data(:,1:D);

            % Initialize parameters
            params = this.initaliseParameters(C,D);
            % Count on number of EM steps
            EM_step = 1;
            % change in likelihood
            lc = 1;
            if (this.Verbose)
                    disp('Training started');
            end
            while(lc > threshold && EM_step <= this.MaxIters)
                likelihood_before = this.computeLikelihood(data,
                   params, D);
                % Generate matrix of posterior probabilties
                H = this.generatePosteriorProbability(data, params
                   , C, D);
                % Update parameters
                params = this.updateParams(data, H, C);
                likelihood_after = this.computeLikelihood(data,
                   params, D);
                % change in likelihood
                lc = abs(likelihood_after - likelihood_before);
                EM_step = EM_step + 1;
            end
            if (this.Verbose)
                    disp('Training ccompleted');
            end
            this.Params = params;
        end

        %
          ____________________________________________________________________

        function p = predict(this, x)
            % Calculate the class conditional probability using
               the learnt
            % GMM and given data point x
            [mean, cov, P] = this.unpackParams(this.Params,this.
               Dimension);
```

```
            p = 0;
            % Predict using first d feature dimension.
            d = length(x);
            for j = 1 :  size(this.Params,1)
                p = p + this.mvg(x, mean(j,1:d), cov(j,1:d))*P(j);
            end
        end
    end

    %
        ------------------------------------------------------------------------
    % Private helper methods
    %
        ------------------------------------------------------------------------
    methods (Access = private)
        %
            --------------------------------------------------------------------
        function params_new = updateParams(this, data, H, C)
            % Update the parameters given the posterior
                probabiities
            % Get current parameters
            N = length(data);

            % Update component probability
            p_new = max(sum(H,1)/N,0.001);
            mean_new = zeros(C,this.Dimension);
            cov_new = zeros(C,this.Dimension);

            for j = 1 : C
                % update component mean
                mean_new(j,:) = sum(H(:,j).*data)/(N*p_new(j));

                % update component covariance
                cov_new(j,:) = max(sum(H(:,j).*power(data-
                    mean_new(j,:),2))/(N*p_new(j)),1e-3);
            end

            % Update parameters
            params_new = this.packParams(mean_new, cov_new, p_new
                ');
        end

        %
            --------------------------------------------------------------------
```

```
        function H = generatePosteriorProbability(this, data, params, C, D)
            % Function generates the posterior probability matrix
            H = zeros(size(data,1), C);
            for i = 1 : size(H,1)
                for j = 1 : size(H,2)
                    [mean, cov, p] = this.unpackParams(params(j,:), D);
                    H(i,j) = this.mvg(data(i,:),mean, cov)*p;
                end
            end
            H = H./sum(H,2);
        end

        % ------------------------------------------------------------------------

        function params = initaliseParameters(this, C, D)
            % Random initialization of parameters of the C components. Each
            % component has a D dimensional mean & covaraince and probability value
            % assosciated with that class
            % M - CxD mean matrix
            % C - CxD covariance matrix
            % P - 1xD probability vector

            % probabilities should add up to 1.
            P = rand(C,1);
            P = P/sum(P);

            % Initialize mean and covaraince.
            M = rand(C,D);
            Co = rand(C,D);
            params = this.packParams(M, Co, P);
        end

        % ------------------------------------------------------------------------

        function likelihood = computeLikelihood(this, data, params, D)
            likelihood = 0;
            [mean, cov,  P] = this.unpackParams(params,D);
            for i = 1 : size(data,1)
                % compute probability of each data point
                p = 0;
                for j = 1 :  size(params,1)
```

```
                    p = p + this.mvg(data(i,:), mean(j,:), cov(j,:))*P(j);
                end
                likelihood = likelihood + log(p);
            end
        end
    end

    methods (Static, Access = private)
        % ----------------------------------------------------------------------

        function [mean, cov, p] = unpackParams(params,D)
            % Helper function to extract parameters
            mean = params(:,1:D);
            cov = params(:,D+1:2*D);
            p = params(:,2*D+1);
        end

        % ----------------------------------------------------------------------

        function params = packParams(mean, cov, p)
            % Pack parameters into 1 unified matrix
            params = [mean cov p];
        end

        % ----------------------------------------------------------------------

        function P = mvg(x, m, c)
            % Computes the probability of x defined by a multivariate gaussian
            % distribution with mean m, covariance c
            d = length(x);
            c = diag(c);
            P = exp(-((x-m)*inv(c)*(x-m)')/2)/(sqrt(power(2*pi,d)*det(c)));
        end
    end
end
```

MATLAB Code - Main Experiment file

```
% Script to run experiments for Q6(a) & Q6(b)

clc;
clear all;

load('TrainingSamplesDCT_8_new.mat')

%% Training params
maxIter = 100;
C = [1,2,4,8,16,32];
d = [1,2,4,8,16,32,40,48,56,64];

if ~(isfile('Q6_a.mat'))
%% Q6 (a)
    disp('Learning Models for Q6(a)')
    models_1 = cell(5,2);
    for i = 1 : 5
        disp(strcat('GMM_' , int2str(i)))
        % learn mixture models with 8 components
        c = 8;
        GMM_FG_64 = GaussianMixtureModel(c, 64, 1e-9, maxIter,true);
        GMM_FG_64.train(TrainsampleDCT_FG);

        GMM_BG_64 = GaussianMixtureModel(c, 64, 1e-9, maxIter,true);
        GMM_BG_64.train(TrainsampleDCT_BG);

        % Save models
        models_1{i,1} = GMM_FG_64;
        models_1{i,2} = GMM_BG_64;
    end
    save('Q6_a','models_1')
else
    load('Q6_a.mat')
end

%% Q6 (b)
if ~(isfile('Q6_b.mat'))
    disp('Learning models for Q6(b)')
    % Train mixture models with different sizes and save them
    models_2 = cell(length(C),2);
    for c_idx = 1 : length(C)
        c = C(c_idx);
        disp(strcat('Learning GMM of c = ', int2str(c)));

```

```
        % learn mixture models with 'c' components
        GMM_FG_64 = GaussianMixtureModel(c, 64, 1e-9, maxIter,true
            );
        GMM_FG_64.train(TrainsampleDCT_FG);

        GMM_BG_64 = GaussianMixtureModel(c, 64, 1e-9, maxIter,true
            );
        GMM_BG_64.train(TrainsampleDCT_BG);

        % Save models
        models_2{c_idx,1} = GMM_FG_64;
        models_2{c_idx,2} = GMM_BG_64;
    end

    save('Q6_b','models_2')
else
    load('Q6_b.mat')
end

%% Set up data
I_cheetah_DCT = getDCTMatrix();
I_maskGT = imread('cheetah_mask.bmp');

% Prior probabilities
pFG = size(TrainsampleDCT_FG,1)/(size(TrainsampleDCT_BG,1)+size(
    TrainsampleDCT_FG,1));
pBG = 1 - pFG;

%% Predict
% Q6 (a) 25 combinations

pair_id = 1;
for a = 1 : 5
    for b = 1 : 5
        % For each of the 25 pairs of models, predict using d dim
            features.
        pError = d*0;
        modelFG = models_1{a,1};
        modelBG = models_1{b,2};
        for dIdx = 1 : length(d)
            disp(strcat('Model #',int2str(pair_id) ,' and ,dim = '
                , int2str(d(dIdx))));
            pError(dIdx) = segmentAndCalulateError(...
                I_cheetah_DCT, I_maskGT, modelBG, modelFG, d(dIdx)
                    , pFG, pBG);
        end

        % Save error plot
```

```
        name = strcat('GMM_', int2str(pair_id), '_6a');
        save(name, "pError");
        pair_id = pair_id + 1;
    end
end

% Q6 (b) 11 combinations
% Predict on cheetah image
for k = 1 : length(models_2)
    disp(strcat('Predicting using GMM of c = ', int2str(C(k))));
    % For each model, segment the test image with differnet dimension of
    % feature space
    modelFG = models_2{k,1};
    modelBG = models_2{k,2};

    d = [1,2,4,8,16,32,40,48,56,64];
    pError = d*0;
    for dIdx = 1 : length(d)
        disp(strcat('Using c = ', int2str(C(k)), 'and dim = ', int2str(d(dIdx))));
        pError(dIdx) = segmentAndCalulateError(...
            I_cheetah_DCT, I_maskGT, modelBG, modelFG, d(dIdx), pFG, pBG);
    end

    % Save error plot
    name = strcat('GMM_', int2str(C(k)),'_Q6b');
    save(name, "pError");
end

%% Helper function
function I_cheetah_DCT = getDCTMatrix()
   % Helper function to get a 64 channel matrix of size of cheetah image
   % with each pixel storing the corresponding DCT coeffecients
   img = imread('cheetah.bmp');
   img = im2double(img);
   % Initialzie DCT matrix of image
   I_cheetah_DCT = zeros(size(img,1),size(img,2),64);
   % Pad image
   img = padarray(img,[7,7],'replicate','post');
   zigZagIdx = readmatrix('Zig-Zag Pattern.txt');
   for i = 1 : 255
       for j = 1 : 270
           block = img(i:i+7, j:j+7);
           dctF = dct2(block);
           fIdx(zigZagIdx(:)+1) = dctF(:);
```

```
            I_cheetah_DCT(i,j,:) = fIdx(:);
        end
    end
end

% ------------------------------------------------------------------------
function pError = segmentAndCalulateError(I_cheetah_DCT, I_maskGT, ...
    modelBG, modelFG, d, pFG, pBG)
    % Function takes input the DCT matrix, GMM for background and
    % foreground. Segments the image outputs the probability of error.
    mask = zeros(size(I_maskGT));
    for i = 1 : 255
        for j = 1 : 270
            f = squeeze(I_cheetah_DCT(i,j,1:d));

            dFG = modelFG.predict(f')*pFG;
            dBG = modelBG.predict(f')*pBG;
            if(dFG > dBG)
                mask(i,j) = 1;
            end
        end
    end
    pError = calculateError(mask, I_maskGT, pBG, pFG);
end

% ------------------------------------------------------------------------
function pError = calculateError(mask, gTruth, pB, pF)
    % Calculate probability of error given grounf truth mask, original
    % mask and class probabilities
    gTruth = im2double(gTruth);
    nCheetah = nnz(gTruth);
    nGrass = nnz(1 - gTruth);
    nMislabeledCheetah = nnz((mask-gTruth)>0);
    nMislabeledGrass = nnz((mask-gTruth)<0);
    pError = nMislabeledGrass/nGrass*pB + nMislabeledCheetah/nCheetah*pF;
end
```