Homework Set Two

ECE 271A

Name : Hsiu-Wen Yen

ID : A59010599

a) From problem 2, we get $\pi_j = \frac{c_j}{n}$.
Therefore, Prior_Probability(cheetah) = Counts(cheetah) / Total Counts.
The result is same as last week.
What I did last week is intuitive for using frequency of cheetah as probability of cheetah, which is same with result of problem 2.

Ans:
P(cheetah) = 0.1919
P(Grass) = 0.8081

```
clear; clc;
% load trainging sample and image
load('TrainingSamplesDCT_8_new.mat');
cheetah = imread('cheetah.bmp');
% calculate prior probabilities of cheetah and grass
pixel_total_count = size(TrainsampleDCT_FG, 1) + size(TrainsampleDCT_BG, 1);
prior_Pcheetah = size(TrainsampleDCT_FG, 1) / pixel_total_count;
prior_Pgrass = size(TrainsampleDCT_BG, 1) / pixel_total_count;
```

Command Window

New to MATLAB? See resources for Getting Started.

```
>> prior_Pcheetah

prior_Pcheetah =

    0.1919

>> prior_Pgrass

prior_Pgrass =

    0.8081
```

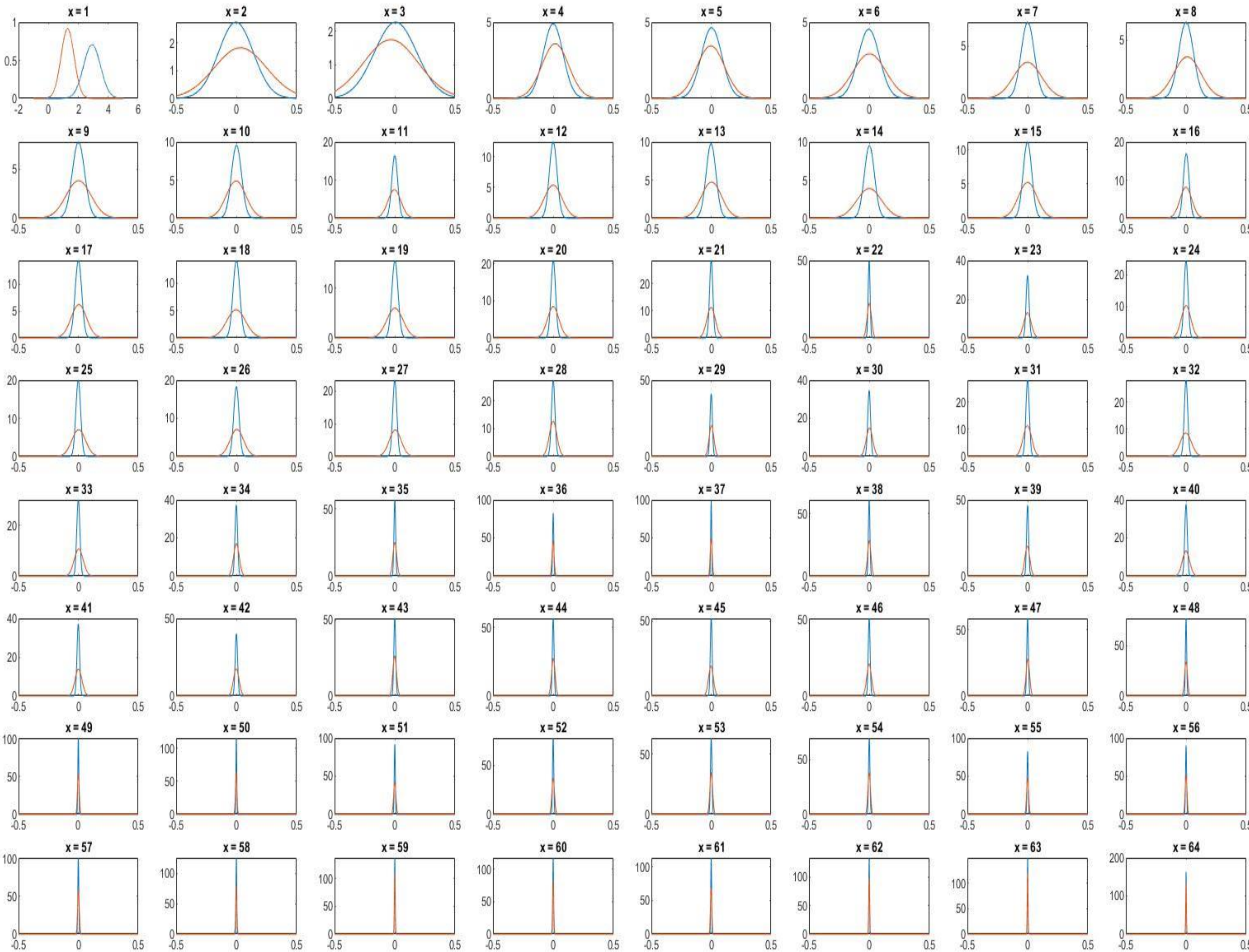
b) First calculate sigma and mean for each dct coefficient.

```
% estimate marginal densities
% front ground
mean_FG = mean(TrainsampleDCT_FG);
variance_FG = var(TrainsampleDCT_FG);
sigma_FG = sqrt(variance_FG);
sum_FG = sum(TrainsampleDCT_FG);
% back ground
mean_BG = mean(TrainsampleDCT_BG);
variance_BG = var(TrainsampleDCT_BG);
sigma_BG = sqrt(variance_BG);
sum_BG = sum(TrainsampleDCT_BG);
```

Plot 64 for marginal densities for the two classes.

```
for i = 1 : 64
    % check negative definite
    % calculate Hessian matrix [A B; B C] for Front ground
    a_FG = - size(TrainsampleDCT_FG,1) / (sigma_FG(i))^2;
    b_FG = -2 * (sum_FG(i) - size(TrainsampleDCT_FG,1) * mean_FG(i)) / (sigma_FG(i))^3;
    c_FG = size(TrainsampleDCT_FG,1) / (sigma_FG(i))^2 - 3 / (sigma_FG(i))^3 * size(TrainsampleDCT_FG,1);
    % calculate Hessian matrix [A B; B C] for Back ground
    a_BG = - size(TrainsampleDCT_BG,1) / (sigma_BG(i))^2;
    b_BG = -2 * (sum_BG(i) - size(TrainsampleDCT_BG,1) * mean_BG(i)) / (sigma_BG(i))^3;
    c_BG = size(TrainsampleDCT_BG,1) / (sigma_BG(i))^2 - 3 / (sigma_BG(i))^3 * size(TrainsampleDCT_BG,1);
    % using eigenvalue of Hessisan matrix to check negative definite
    H = [a_FG, b_FG; b_FG, c_FG];
    e = eig(H);
    if  all(e > 0) % if all eigenvalue is negative, Hessian matrix is negative definite
        continue; % skip plot when one of eigenvalue is positive
    end
    H = [a_BG, b_BG; b_BG, c_BG];
    e = eig(H);
    if  all(e > 0) % if all eigenvalue is negative, Hessian matrix is negative definite
        continue; % skip plot when one of eigenvalue is positive
    end

    % plot BG
    % **NOTE** for first dct coefficient, mean is bigger than other.
    % Therefore, it has different x interval.
    if (i == 1)
        x = -1:0.001:5;
    else
        x = -0.5:0.001:0.5;
    end
    y = 1 / (sigma_BG(i) * sqrt(2 * pi)) * exp(- 0.5 * ((x - mean_BG(i)) / sigma_BG(i)).^2);
    subplot(8, 8, i);
    txt = "x = " + int2str(i);
    plot(x,y);
    hold on;
    % plot FG
    y = 1 / (sigma_FG(i) * sqrt(2 * pi)) * exp(- 0.5 * ((x - mean_FG(i)) / sigma_FG(i)).^2);
    plot(x,y);
    title(txt);
end
```

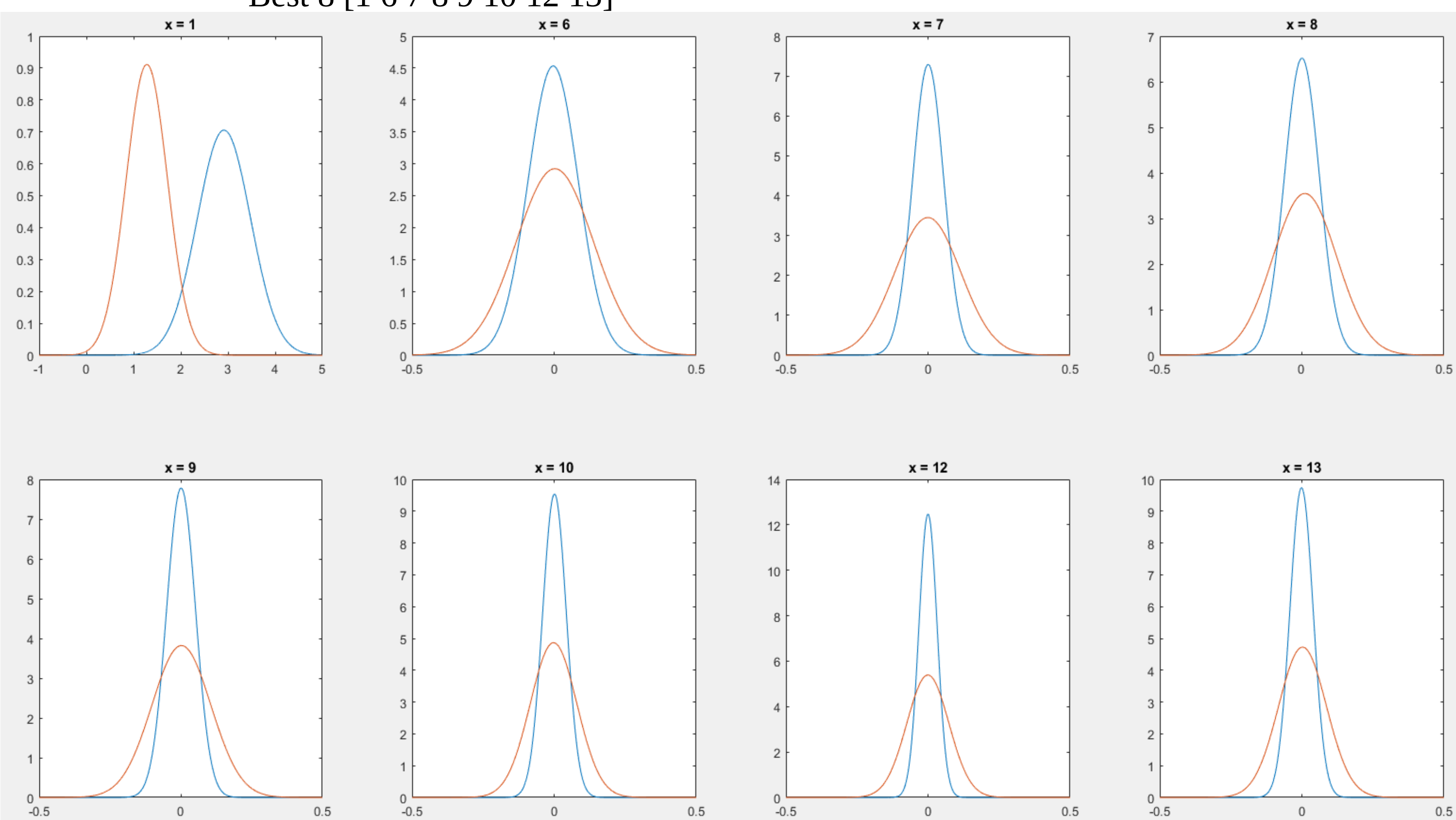

x = 1
x = 2
x = 3
x = 4
x = 5
x = 6
x = 7
x = 8
x = 9
x = 10
x = 11
x = 12
x = 13
x = 14
x = 15
x = 16
x = 17
x = 18
x = 19
x = 20
x = 21
x = 22
x = 23
x = 24
x = 25
x = 26
x = 27
x = 28
x = 29
x = 30
x = 31
x = 32
x = 33
x = 34
x = 35
x = 36
x = 37
x = 38
x = 39
x = 40
x = 41
x = 42
x = 43
x = 44
x = 45
x = 46
x = 47
x = 48
x = 49
x = 50
x = 51
x = 52
x = 53
x = 54
x = 55
x = 56
x = 57
x = 58
x = 59
x = 60
x = 61
x = 62
x = 63
x = 64

From picture above, I think x = [1 6 7 8 9 10 12 13] have clear clarification, you can see two curves distinctly. For x = [2 3 4 5], two curves almost overlap each other.

On the other hand, x = [57 58 59 60 61 62 63 64] have worst clarification, the range for identification is so small.

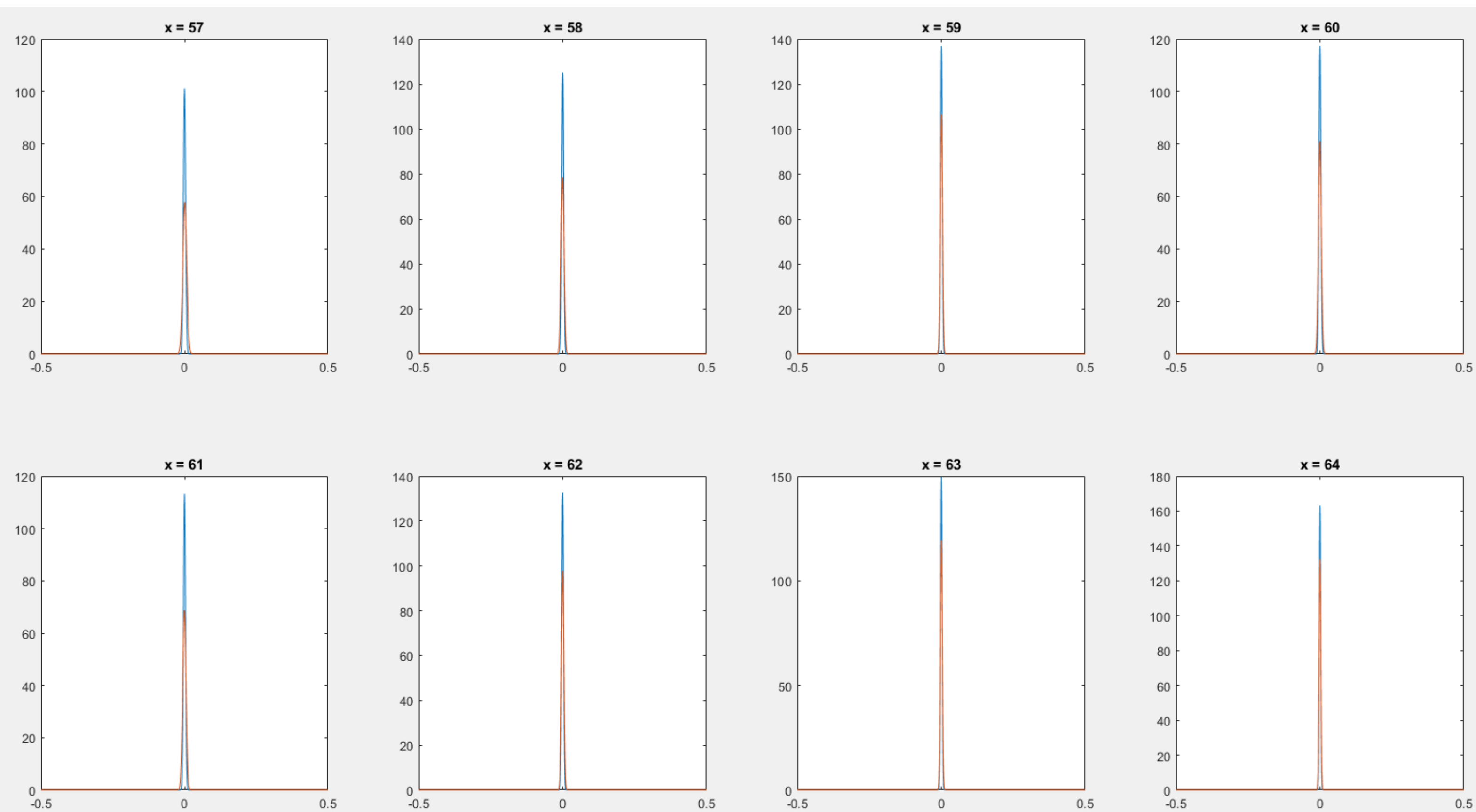
```
%% plot best and worst section
best_idx = [1 6 7 8 9 10 12 13];
worst_idx = [57, 58, 59, 60, 61, 62, 63, 64];
% plot best
figure(2);
for counter = 1 : size(best_idx, 2)
    i = best_idx(counter);
    % plot BG
    if (i == 1)
        x = -1:0.001:5;
    else
        x = -0.5:0.001:0.5;
    end
    y = 1 / (sigma_BG(i) * sqrt(2 * pi)) * exp(- 0.5 * ((x - mean_BG(i)) / sigma_BG(i)).^2);
    subplot(2, 4, counter);
    txt = "x = " + int2str(i);
    plot(x,y);
    hold on;
    % plot FG
    y = 1 / (sigma_FG(i) * sqrt(2 * pi)) * exp(- 0.5 * ((x - mean_FG(i)) / sigma_FG(i)).^2);
    plot(x,y);
    title(txt);
end
```

Best 8 [1 6 7 8 9 10 12 13]



```
% plot worst
figure(3);
for counter = 1 : size(worst_idx, 2)
    i = worst_idx(counter);
    % plot BG
    if (i == 1)
        x = -1:0.001:5;
    else
        x = -0.5:0.001:0.5;
    end
    y = 1 / (sigma_BG(i) * sqrt(2 * pi)) * exp(- 0.5 * ((x - mean_BG(i)) / sigma_BG(i)).^2);
    subplot(2, 4, counter);
    txt = "x = " + int2str(i);
    plot(x,y);
    hold on;
    % plot FG
    y = 1 / (sigma_FG(i) * sqrt(2 * pi)) * exp(- 0.5 * ((x - mean_FG(i)) / sigma_FG(i)).^2);
    plot(x,y);
    title(txt);
end
```

Worst 8 [57 58 59 60 61 62 63 64]



c) First calculate 64D covariance

```
%% 64D feature section
% calculate covariance for 64D for FG
covariance_64_FG = cov(TrainsampleDCT_FG);
covariance_64_BG = cov(TrainsampleDCT_BG);

% create output mask image array
row_size = size(cheetah, 1);
column_size = size(cheetah, 2);
A_64 = zeros(row_size, column_size);
```

According BDR to create cheetah mask for 64D.

```
% using 8 * 8 blocks to represent the left top pixel
for rows = 1 : row_size - 8 + 1
    for columns = 1 : column_size - 8 + 1
        block = cheetah(rows:rows+7, columns:columns+7);
        block = dct2(block);
        x = expand_zigzag(block);
        % for FG and BG
        p_FG = (-0.5*(x - mean_FG)/ covariance_64_FG * (x - mean_FG).') - log(sqrt(det(covariance_64_FG)*(2*pi)^64)) + log(prior_Pcheetah);

        p_BG = (-0.5*(x - mean_BG)/ covariance_64_BG * (x - mean_BG).') - log(sqrt(det(covariance_64_BG)*(2*pi)^64)) + log(prior_Pgrass);

        if (p_BG > p_FG)
            A_64(rows, columns) = 0;
        else
            A_64(rows, columns) = 1;
        end
    end
end
figure(4);
imagesc(A_64);
colormap(gray(255));
```

For best 8d feature, do the same thing.

Calculate 8d covariance and mean.

```
%% 8D feature section
% extrac required feature from training set
for j = 1 : 8
    required_8d_FG(:,j) = TrainsampleDCT_FG(:, best_idx(j));
    required_8d_BG(:,j) = TrainsampleDCT_BG(:, best_idx(j));
end

% calculate covariance and mean for 8D
covariance_8_FG = cov(required_8d_FG);
covariance_8_BG = cov(required_8d_BG);
mean_8d_FG = mean(required_8d_FG);
mean_8d_BG = mean(required_8d_BG);
```

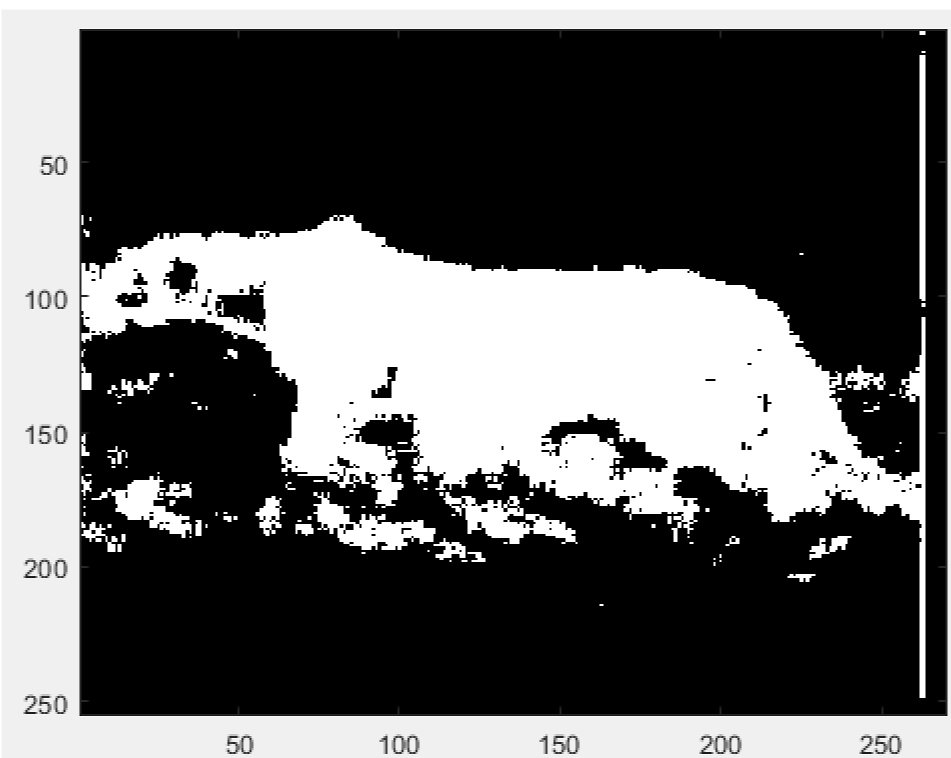
According BDR to create cheetah mask for 8D.

```
A_8 = zeros(row_size, column_size);
% using 8 * 8 blocks to represent the left top pixel
for rows = 1 : row_size - 8 + 1
    for columns = 1 : column_size - 8 + 1
        block = cheetah(rows:rows+7, columns:columns+7);
        block = dct2(block);
        x = expand_zigzag(block);
        % for FG and BG
        % get feature value
        for j = 1 : 8
            x_8d(j) = x(best_idx(j));
        end
        p_FG = (-0.5*(x_8d - mean_8d_FG)/ covariance_8_FG * (x_8d - mean_8d_FG).') - log(sqrt(det(covariance_8_FG)*(2*pi)^64)) + log(prior_Pcheetah);
        p_BG = (-0.5*(x_8d - mean_8d_BG)/ covariance_8_BG * (x_8d - mean_8d_BG).') - log(sqrt(det(covariance_8_BG)*(2*pi)^64)) + log(prior_Pgrass);
        if (p_BG > p_FG)
            A_8(rows, columns) = 0;
        else
            A_8(rows, columns) = 1;
        end
    end
end
figure(5);
imagesc(A_8);
colormap(gray(255));
```

For estimating probability of error, we need to count how many pixels aren't correct, and then divide it with total pixel counts.

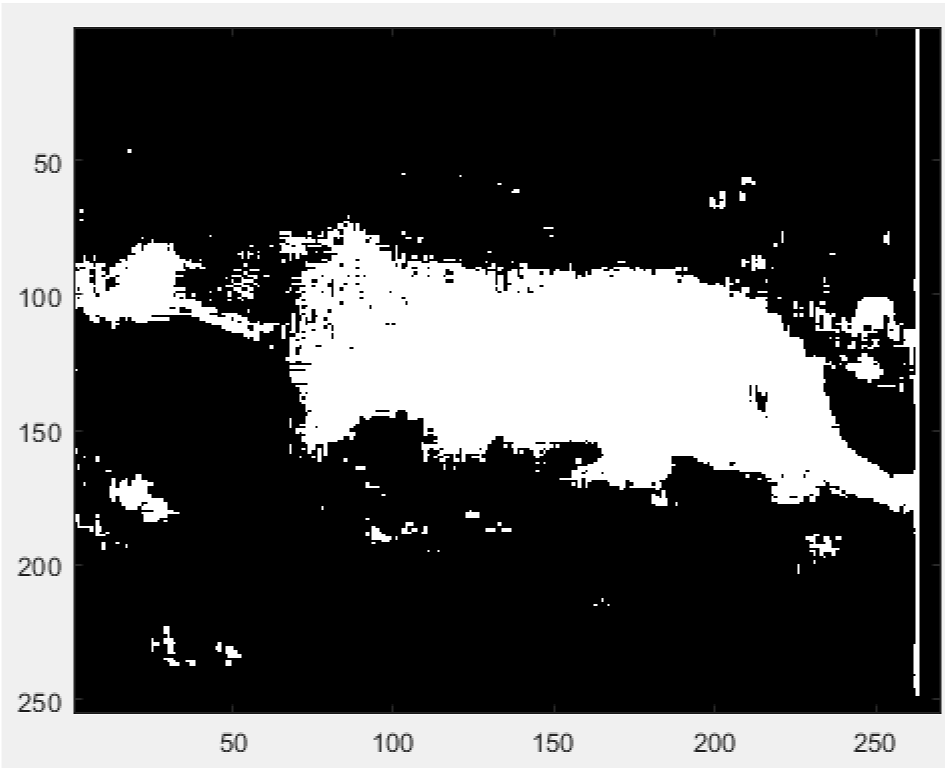
```
%% error
% load cheetah mask.bmp
truth = imread("cheetah_mask.bmp");
% calculate last meaningful index of row and column
last_row = size(cheetah, 1) - 8 + 1;
last_column = size(cheetah, 2) - 8 + 1;
% error for 64d
truth = double(truth(1 : last_row, 1 : last_column) / 255);
A_64 = A_64(1 : last_row, 1 : last_column);
err = truth - A_64;
err = abs(err);
probability_error_64d = sum(err,'all') / (last_row*last_column);
% error for 8d
A_8 = A_8(1 : last_row, 1 : last_column);
err = truth - A_8;
err = abs(err);
probability_error_8d = sum(err,'all') / (last_row*last_column);
```

64d Result:

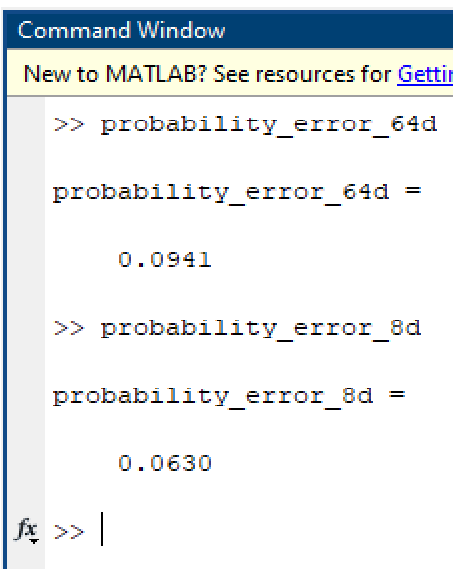


Error : 0.094060468539188

8d Result:



Error : 0.063013614620385



Result of 64d is worse than 8d because 64d contains too many meaningless features that influence the decision rule. On the other hand, 8d only took useful features for calculating probability, therefore, result is much better than 8d.

I think it is kind of overfitting situation for 64d.

Full code review:

```
clear; clc;
% load trainging sample and image
load('TrainingSamplesDCT_8_new.mat');
cheetah = imread('cheetah.bmp');
cheetah=double(cheetah)/255;
% calculate prior probabilities of cheetah and grass
pixel_total_count = size(TrainsampleDCT_FG, 1) + size(TrainsampleDCT_BG, 1);
prior_Pcheetah = size(TrainsampleDCT_FG, 1) / pixel_total_count;
prior_Pgrass = size(TrainsampleDCT_BG, 1) / pixel_total_count;

% estimate marginal densities
% front ground
mean_FG = mean(TrainsampleDCT_FG);
variance_FG = var(TrainsampleDCT_FG);
sigma_FG = sqrt(variance_FG);
sum_FG = sum(TrainsampleDCT_FG);
% back ground
mean_BG = mean(TrainsampleDCT_BG);
variance_BG = var(TrainsampleDCT_BG);
sigma_BG = sqrt(variance_BG);
sum_BG = sum(TrainsampleDCT_BG);

figure(1);
for i = 1 : 64
    % check negative definite
    % calculate Hessian matrix [A B; B C] for Front ground
    a_FG = - size(TrainsampleDCT_FG,1) / (sigma_FG(i))^2;
    b_FG = -2 * (sum_FG(i) - size(TrainsampleDCT_FG,1) * mean_FG(i)) /
(sigma_FG(i))^3;
    c_FG = size(TrainsampleDCT_FG,1) / (sigma_FG(i))^2 - 3 / (sigma_FG(i))^3 *
size(TrainsampleDCT_FG,1);
    % calculate Hessian matrix [A B; B C] for Back ground
    a_BG = - size(TrainsampleDCT_BG,1) / (sigma_BG(i))^2;
    b_BG = -2 * (sum_BG(i) - size(TrainsampleDCT_BG,1) * mean_BG(i)) /
(sigma_BG(i))^3;
    c_BG = size(TrainsampleDCT_BG,1) / (sigma_BG(i))^2 - 3 / (sigma_BG(i))^3 *
size(TrainsampleDCT_BG,1);
    % using eigenvalue of Hessisan matrix to check negative definite
    H = [a_FG, b_FG; b_FG, c_FG];
    e = eig(H);
    if  all(e > 0) % if all eigenvalue is negative, Hessian matrix is negative
definite
        continue; % skip plot when one of eigenvalue is positive
    end
    H = [a_BG, b_BG; b_BG, c_BG];
    e = eig(H);
    if  all(e > 0) % if all eigenvalue is negative, Hessian matrix is negative
definite
        continue; % skip plot when one of eigenvalue is positive
    end

    % plot BG
    % **NOTE** for first dct coefficient, mean is bigger than other.
```

```
    % Therefore, it has different x interval.
    if (i == 1)
        x = -1:0.001:5;
    else
        x = -0.5:0.001:0.5;
    end
    y = 1 / (sigma_BG(i) * sqrt(2 * pi)) * exp(- 0.5 * ((x - mean_BG(i)) /
sigma_BG(i)).^2);
    subplot(8, 8, i);
    txt = "x = " + int2str(i);
    plot(x,y);
    hold on;
    % plot FG
    y = 1 / (sigma_FG(i) * sqrt(2 * pi)) * exp(- 0.5 * ((x - mean_FG(i)) /
sigma_FG(i)).^2);
    plot(x,y);
    title(txt);
end

%% plot best and worst section
best_idx = [1 6 7 8 9 10 12 13];
worst_idx = [57, 58, 59, 60, 61, 62, 63, 64];
% plot best
figure(2);
for counter = 1 : size(best_idx, 2)
    i = best_idx(counter);
    % plot BG
    if (i == 1)
        x = -1:0.001:5;
    else
        x = -0.5:0.001:0.5;
    end
    y = 1 / (sigma_BG(i) * sqrt(2 * pi)) * exp(- 0.5 * ((x - mean_BG(i)) /
sigma_BG(i)).^2);
    subplot(2, 4, counter);
    txt = "x = " + int2str(i);
    plot(x,y);
    hold on;
    % plot FG
    y = 1 / (sigma_FG(i) * sqrt(2 * pi)) * exp(- 0.5 * ((x - mean_FG(i)) /
sigma_FG(i)).^2);
    plot(x,y);
    title(txt);
end
% plot worst
figure(3);
for counter = 1 : size(worst_idx, 2)
    i = worst_idx(counter);
    % plot BG
    if (i == 1)
        x = -1:0.001:5;
    else
        x = -0.5:0.001:0.5;
    end
```

```
    y = 1 / (sigma_BG(i) * sqrt(2 * pi)) * exp(- 0.5 * ((x - mean_BG(i)) /
sigma_BG(i)).^2);
    subplot(2, 4, counter);
    txt = "x = " + int2str(i);
    plot(x,y);
    hold on;
    % plot FG
    y = 1 / (sigma_FG(i) * sqrt(2 * pi)) * exp(- 0.5 * ((x - mean_FG(i)) /
sigma_FG(i)).^2);
    plot(x,y);
    title(txt);
end

%% 64D feature section
% calculate covariance for 64D for FG
covariance_64_FG = cov(TrainsampleDCT_FG);
covariance_64_BG = cov(TrainsampleDCT_BG);

% create output mask image array
row_size = size(cheetah, 1);
column_size = size(cheetah, 2);
A_64 = zeros(row_size, column_size);

% using 8 * 8 blocks to represent the left top pixel
for rows = 1 : row_size - 8 + 1
    for columns = 1 : column_size - 8 + 1
        block = cheetah(rows:rows+7, columns:columns+7);
        block = dct2(block);
        x = expand_zigzag(block);
        % for FG and BG
        p_FG = (-0.5*(x - mean_FG)/ covariance_64_FG * (x - mean_FG).') -
log(sqrt(det(covariance_64_FG)*(2*pi)^64)) + log(prior_Pcheetah);

        p_BG = (-0.5*(x - mean_BG)/ covariance_64_BG * (x - mean_BG).') -
log(sqrt(det(covariance_64_BG)*(2*pi)^64)) + log(prior_Pgrass);

        if (p_BG > p_FG)
            A_64(rows, columns) = 0;
        else
            A_64(rows, columns) = 1;
        end
    end
end
figure(4);
imagesc(A_64);
colormap(gray(255));
%% 8D feature section
% extrac required feature from training set
for j = 1 : 8
    required_8d_FG(:,j) = TrainsampleDCT_FG(:, best_idx(j));
    required_8d_BG(:,j) = TrainsampleDCT_BG(:, best_idx(j));
end

% calculate covariance and mean for 8D
covariance_8_FG = cov(required_8d_FG);
```

```
covariance_8_BG = cov(required_8d_BG);
mean_8d_FG = mean(required_8d_FG);
mean_8d_BG = mean(required_8d_BG);

A_8 = zeros(row_size, column_size);
% using 8 * 8 blocks to represent the left top pixel
for rows = 1 : row_size - 8 + 1
    for columns = 1 : column_size - 8 + 1
        block = cheetah(rows:rows+7, columns:columns+7);
        block = dct2(block);
        x = expand_zigzag(block);
        % for FG and BG
        % get feature value
        for j = 1 : 8
            x_8d(j) = x(best_idx(j));
        end
        p_FG = (-0.5*(x_8d - mean_8d_FG)/ covariance_8_FG * (x_8d - mean_8d_FG).') -
log(sqrt(det(covariance_8_FG)*(2*pi)^64)) + log(prior_Pcheetah);
        p_BG = (-0.5*(x_8d - mean_8d_BG)/ covariance_8_BG * (x_8d - mean_8d_BG).') -
log(sqrt(det(covariance_8_BG)*(2*pi)^64)) + log(prior_Pgrass);
        if (p_BG > p_FG)
            A_8(rows, columns) = 0;
        else
            A_8(rows, columns) = 1;
        end
    end
end
figure(5);
imagesc(A_8);
colormap(gray(255));

%% error
% load cheetah mask.bmp
truth = imread("cheetah_mask.bmp");
% calculate last meaningful index of row and column
last_row = size(cheetah, 1) - 8 + 1;
last_column = size(cheetah, 2) - 8 + 1;
% error for 64d
truth = double(truth(1 : last_row, 1 : last_column) / 255);
A_64 = A_64(1 : last_row, 1 : last_column);
err = truth - A_64;
err = abs(err);
probability_error_64d = sum(err,'all') / (last_row*last_column);
% error for 8d
A_8 = A_8(1 : last_row, 1 : last_column);
err = truth - A_8;
err = abs(err);
probability_error_8d = sum(err,'all') / (last_row*last_column);
```

```
function myArray = expand_zigzag(matrix)
    load("Zig-Zag Pattern.txt");
    myArray = zeros(1, 64);
    for row = 1 : size(matrix,1)
        for column = 1 : size(matrix,2)
```

```
            number = Zig_Zag_Pattern(row, column) + 1;
            myArray(number) = matrix(row, column);
        end
    end
end
```