Jiaming Lai
A53314574

Assignment #3

Computer Problem Solution

a) sdf

Appendix

The following is the Matlab code.

0.1 HW3_solution.m

```
clear all
%Training
%Read the TrainingSamplesDCT_8.mat file
load('dataset/TrainingSamplesDCT_subsets_8.mat');
load('dataset/Alpha.mat');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Strategy 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Use strategy 1 and train set D1
%Save D1_BG and D1_FG in temporary value
train_BG = D1_BG;
train_FG = D1_FG;
fun_general(1,train_BG,train_FG,alpha,1);
%Use strategy 1 and train set D2
%Save D1_BG and D1_FG in temporary value
train_BG = D2_BG;
train_FG = D2_FG;
fun_general(2,train_BG,train_FG,alpha,1);
%Use strategy 1 and train set D3
%Save D1_BG and D1_FG in temporary value
train_BG = D3_BG;
train_FG = D3_FG;
fun_general(3,train_BG,train_FG,alpha,1);
%Use strategy 1 and train set D4
%Save D1_BG and D1_FG in temporary value
train_BG = D4_BG;
train_FG = D4_FG;
fun_general(4,train_BG,train_FG,alpha,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Strategy 2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Use strategy 2 and train set D1
%Save D1_BG and D1_FG in temporary value
train_BG = D1_BG;
train_FG = D1_FG;
```

```
fun_general(1,train_BG,train_FG,alpha,2);
%Use strategy 1 and train set D2
%Save D1_BG and D1_FG in temporary value
train_BG = D2_BG;
train_FG = D2_FG;
fun_general(2,train_BG,train_FG,alpha,2);
%Use strategy 1 and train set D3
%Save D1_BG and D1_FG in temporary value
train_BG = D3_BG;
train_FG = D3_FG;
fun_general(3,train_BG,train_FG,alpha,2);
%Use strategy 1 and train set D4
%Save D1_BG and D1_FG in temporary value
train_BG = D4_BG;
train_FG = D4_FG;
fun_general(4,train_BG,train_FG,alpha,2);
```

0.2 fun_general.m

This function receives parameters: train data, array of alpha and strategy we want to choose, and compute error using MAP-BDR, Bayes-BDR and ML-BDR.

```
function []=fun_general(dataset,train_BG,train_FG,alpha,i)
%This function is comparasion of using different strategy, different
   dataset and
%different method plugging in BDR
%i - represent the strategy we use

%Define the strategy we use
s=i; k=1;
%Define the array to store error
error_pre = zeros(size(alpha));
error_ml = zeros(size(alpha));
error_map = zeros(size(alpha));

% for Alpha=alpha
%     error_map(1,k) = fun_mapBDR(train_BG,train_FG,Alpha,s)
%     k=k+1;
% end

for Alpha=alpha
    error_pre(1,k) = fun_bayesBDR(train_BG,train_FG,Alpha,s)
    error_map(1,k) = fun_mapBDR(train_BG,train_FG,Alpha,s)
    k=k+1;
end
error_ml(1,:)=fun_mlBDR(train_BG,train_FG);
%Plot the data
figure(1);
line_pre = plot(alpha,error_pre);
hold on;
line_map = plot(alpha,error_map);
```

```
hold on;
line_ml = plot(alpha,error_ml);
legend({'Predictive','MAP','ML'},'Location','southeast');
set(gca,'XScale','log');
grid on;
title({['Dataset_',num2str(dataset),' and Strategy_',num2str(i)]...
    ;['PoE vs Alpha']},'Fontsize',12,'interpreter','latex');
ylabel('PoE', 'interpreter', 'latex');
xlabel('Alpha', 'interpreter', 'latex');
set(gcf,'Position',[400,100,900,600]);
saveas(gcf, ['images/PoE of D',num2str(dataset),' and strategy'...
   , num2str(i),'.jpg']);
close(gcf);

end
```

0.3 fun_bayesBDR.m

This function performs Bayes-BDR.

```
function [error] = fun_bayesBDR(train_BG,train_FG,Alpha,i)
%This function is for Bayes BDR.
%Use Bayes Estimation and BDR, return the predict mask of original image.
%i - represent the strategy we use

%Load data
load(['dataset/Prior_',num2str(i),'.mat']);
%Read the mask file
I = imread('dataset/cheetah_mask.bmp');
I = im2double(I);
%Define the predict mask
mask_64 = zeros(size(I));

P_BG = size(train_BG,1) / (size(train_BG,1) + size(train_FG,1));
P_FG = size(train_FG,1) / (size(train_BG,1) + size(train_FG,1));
%Calculate the mean of every features when cheetah
%mean_ch is the mean
mean_ch = fun_mean(train_FG);
%Calculate the mean of every features when grass
%mean_gr is the mean
mean_gr = fun_mean(train_BG);
%Compute the covariance matrix of class-condition
cov_ch = fun_cov(train_FG,mean_ch);
cov_gr = fun_cov(train_BG,mean_gr);
%Compute the covariance matrix of Gaussian prior
v = Alpha * W0;
cov_prior = diag(v);
% size(cov_prior)
% size(cov_ch)
% size(cov_gr)
% size(mu0_BG)

%Compute Bayes Estimation and parameters of the predictive distribution
%Use mu_p and mu_cov as the mean and convariance matrix of the predictive
%distribution. The equation is from DHS.
%BG predictive distribution
mu_p_BG = cov_prior * inv(cov_prior + cov_gr/size(train_BG,1)) * ...
    mean_gr' + cov_gr/size(train_BG,1) * inv(cov_prior + cov_gr/size(
   train_BG,1))...
    * mu0_BG';
cov_p_BG = cov_gr + cov_prior * inv(cov_prior + cov_gr/size(train_BG,1)) *
     cov_gr/size(train_BG,1);
%FG predictive distribution
mu_p_FG = cov_prior * inv(cov_prior + cov_ch/size(train_FG,1)) * ...
    mean_ch' + cov_ch/size(train_FG,1) * inv(cov_prior + cov_ch/size(
   train_FG,1))...
    * mu0_FG';
cov_p_FG = cov_ch + cov_prior * inv(cov_prior + cov_ch/size(train_FG,1)) *
     cov_ch/size(train_FG,1);

%Load DCT file
load('DCT_coeffience.mat');
%Caculate the threshold
T = P_BG / P_FG;
%Define the loop numbers
loop_row = size(I,1) - 8 + 1;
loop_column = size(I,2) - 8 + 1;
k=1;

for i=1:1:loop_row
    for j=1:1:loop_column
        P_x_FG = fun_mvgaussian(DCT_coeffience(k,:),mu_p_FG',cov_p_FG);
        P_x_BG = fun_mvgaussian(DCT_coeffience(k,:),mu_p_BG',cov_p_BG);
        if P_x_FG/P_x_BG > T
            mask_64(i,j) = 1;
        end
        k=k+1;
    end
end
%Calculate the probability of error
error = length(find((mask_64-I)~=0)) / (size(I,1) * size(I,2));

end
```

0.4 fun_mapBDR.m

This function performs MAP-BDR.

```
function [error] = fun_mapBDR(train_BG,train_FG,Alpha,i)
%This function is for MAP BDR.
%Use Bayes Estimation and BDR, return the predict mask of original image.
%i - represent the strategy we use

%Load data
load(['dataset/Prior_',num2str(i),'.mat']);
%Read the mask file
I = imread('dataset/cheetah_mask.bmp');
I = im2double(I);
%Define the predict mask
mask_64 = zeros(size(I));

P_BG = size(train_BG,1) / (size(train_BG,1) + size(train_FG,1));
P_FG = size(train_FG,1) / (size(train_BG,1) + size(train_FG,1));
%Calculate the mean of every features when cheetah
%mean_ch is the mean
mean_ch = fun_mean(train_FG);
%Calculate the mean of every features when grass
%mean_gr is the mean
mean_gr = fun_mean(train_BG);
%Compute the covariance matrix of class-condition
cov_ch = fun_cov(train_FG,mean_ch);
cov_gr = fun_cov(train_BG,mean_gr);
%Compute the covariance matrix of Gaussian prior
v = Alpha * W0;
cov_prior = diag(v);

%Compute MAP Estimation and parameters of the predictive distribution
%Use mu_p and mu_cov as the mean and convariance matrix of the predictive
%distribution. The equation is from DHS.
%BG predictive distribution
mu_p_BG = cov_prior * inv(cov_prior + cov_gr/size(train_BG,1)) * ...
    mean_gr' + cov_gr/size(train_BG,1) * inv(cov_prior + cov_gr/size(
   train_BG,1))...
    * mu0_BG';
cov_p_BG = cov_gr;
%FG predictive distribution
mu_p_FG = cov_prior * inv(cov_prior + cov_ch/size(train_FG,1)) * ...
    mean_ch' + cov_ch/size(train_FG,1) * inv(cov_prior + cov_ch/size(
   train_FG,1))...
    * mu0_FG';
cov_p_FG = cov_ch;

%Load DCT file
load('DCT_coeffience.mat');
%Caculate the threshold
T = P_BG / P_FG;
%Define the loop numbers
loop_row = size(I,1) - 8 + 1;
```

```
loop_column = size(I,2) - 8 + 1;
k=1;

for i=1:1:loop_row
    for j=1:1:loop_column
        P_x_FG = fun_mvgaussian(DCT_coeffience(k,:),mu_p_FG',cov_p_FG);
        P_x_BG = fun_mvgaussian(DCT_coeffience(k,:),mu_p_BG',cov_p_BG);
        if P_x_FG/P_x_BG > T
            mask_64(i,j) = 1;
        end
        k=k+1;
    end
end
%Calculate the probability of error
error = length(find((mask_64-I)~=0)) / (size(I,1) * size(I,2));

end
```

0.5 fun_mlBDR.m

This function performs ML-BDR.

```
function [error] = fun_bayesBDR(train_BG,train_FG)
%This function is for ML BDR.

%Read the mask file
I = imread('dataset/cheetah_mask.bmp');
I = im2double(I);
%Define the predict mask
mask_64 = zeros(size(I));

P_BG = size(train_BG,1) / (size(train_BG,1) + size(train_FG,1));
P_FG = size(train_FG,1) / (size(train_BG,1) + size(train_FG,1));
%Calculate the mean of every features when cheetah
%mean_ch is the mean
mean_ch = fun_mean(train_FG);
%Calculate the mean of every features when grass
%mean_gr is the mean
mean_gr = fun_mean(train_BG);

%Calculate the covariance matrix of cheetah
cov_ch = fun_cov(train_FG,mean_ch);
%Calculate the covariance matrix of grass
cov_gr = fun_cov(train_BG,mean_gr);

%Load DCT file
load('DCT_coeffience.mat');
%Caculate the threshold
T = P_BG / P_FG;
```

```
%Define the loop numbers
loop_row = size(I,1) - 8 + 1;
loop_column = size(I,2) - 8 + 1;
k=1;

for z=1:1:loop_row
    for j=1:1:loop_column
        P_x_FG = fun_mvgaussian(DCT_coeffience(k,:),mean_ch,cov_ch);
        P_x_BG = fun_mvgaussian(DCT_coeffience(k,:),mean_gr,cov_gr);
        if P_x_FG/P_x_BG > T
            mask_64(z,j) = 1;
        end
        k=k+1;
    end
end
%Calculate the probability of error
error = length(find((mask_64-I)~=0)) / (size(I,1) * size(I,2));

end
```

0.6 fun_zigzag.m

This function compute DCT coefficients and save it as DCT_coefficience.mat.

```
function [] = fun_zigzag()
%This function is for computing the DCT coe?cients of original image.
%Decomposition into 8 x 8 image blocks, compute the DCT of each block, and
%zig-zag scan.

%Read original image
I = imread('dataset/cheetah.bmp');
I = im2double(I);
%Define the loop numbers
loop_row = size(I,1) - 8 + 1;
loop_column = size(I,2) - 8 + 1;
%Read the Zig-Zag file
position_ref = load('dataset/Zig-Zag Pattern.txt');
%Define the array for saving DCT coeffiences according to Zig-Zag
DCT_coeffience = zeros([loop_row*loop_column,64]);
k=1;

for i=1:1:loop_row
    for j=1:1:loop_column
        block = I(i:i+7,j:j+7);
        DCT_block = dct2(block);
        %Map DCT_block matrix to array according Zig-Zag
        for row=1:1:8
            for column=1:1:8
```

```
                DCT_coeffience(k,position_ref(row,column)+1)=DCT_block(row
    ,column);
            end
        end
        k=k+1;
    end
end
save('../DCT_coeffience.mat','DCT_coeffience');
end
```

0.7 fun_mvgaussian.m

This function perfroms multi-gaussian.

```
function [y] = fun_mvgaussian(x,mean,cov)
%This function is for MV Gaussian

d = size(x,2);
y = 1/sqrt((2*pi)^d*det(cov))*exp(-(x-mean)*cov^-1*(x-mean)'/2);

end
```

0.8 fun_cov.m

```
function [cov] = fun_cov(data,data_mean)
%This function is the maximum likelihood estimation of covariance matrix

N = size(data,1);
cov = (data-data_mean)'*(data-data_mean)./N;

end
```

0.9 fun_mean.m

```
function [mean] = fun_mean(data)
%This function is the maximum likelihood estimation of mean

N = size(data,1);
A = ones(1,N);
mean = A * data ./ N;

end
```