Lund University
Faculty of Engineering
Name: ZHANG Yifei
Email: yi4840zh-s@lu.se
Date: November 9, 2023

FMAN20 - Image Analysis, Autumn 2023

Assignment 4

Answer to Task 1 Color correction of images

I convert the image to white world and gray world. The I used WB_sRGB to translate correct image to implement the white balance image.And I calculate the PSNR, SSIM and LIP saparatly.

The code of the demo of WB_sRGB is performed:

```
      %% input and options
infileName = fullfile('..','example_images','michelangelo_colorshift.jpg');
outfileName = fullfile('result.jpg');
device = 'cpu'; % 'cpu' or 'gpu'
gamut_mapping = 2; % use 1 for scaling, 2 for clipping (our paper's results ...
    reported using clipping).
upgraded_model = 1; % use 1 to load our new model that is upgraded with new ...
    training examples.


%%
switch lower(device)
    case 'cpu'
        if upgraded_model == 1
            load(fullfile('models','WB_model+.mat'));
        elseif upgraded_model == 0
            load(fullfile('models','WB_model.mat'));
        else
            error('Wrong upgraded_model value; please use 0 or 1');
        end
    case 'gpu'
        try
            gpuDevice();
        catch
            error('Cannot find a GPU device');
        end
        if upgraded_model == 1
            load(fullfile('models','WB_model+_gpu.mat'));
        elseif upgraded_model == 0
            load(fullfile('models','WB_model_gpu.mat'));
        else
            error('Wrong upgraded_model value; please use 0 or 1');
        end
    otherwise
        error('Wrong device; please use ''gpu'' or ''cpu''')
end
model.gamut_mapping = gamut_mapping;
fprintf('Processing image: %s\n',infileName);
I_in = imread(infileName);
tic
I_corr = model.correctImage(I_in);
disp('Done!');
toc
subplot(1,2,1); imshow(I_in); title('Input');
subplot(1,2,2); imshow(I_corr); title('Our result');
disp('Saving...');
if strcmpi(device,'cpu')
    imwrite(I_corr,outfileName);
else
    imwrite(gather(I_corr),outfileName);
end
disp('Saved!');
```

Gray World Assumption Method:

Calculate the average values of the three channels (R, G, B) of the original image to obtain the average grayscale value.

Multiply the pixel values of each channel by a scaling factor to make the average values of the three channels equal, achieving grayscale balance.

White World Assumption Method:

Calculate the maximum values of each channel in the original image.

Multiply the pixel values of each channel by a scaling factor to make the maximum values of the three channels equal, achieving white balance.

By displaying the original image, the Gray World corrected image, and the White World corrected image, we can intuitively compare the correction effects. Additionally, evaluation metrics such as PSNR (Peak Signal-to-Noise Ratio), SSIM (Structural Similarity Index), and LIP-error (Color Difference) were calculated to assess the correction performance compared to the reference image.

The code of the calculation of the white world and gray world white world and gray world and PSNR, SSIM and LIP is performed:

```
    clc
    clear all;
    close all;

    % Load the image
    image = imread('michelangelo_colorshift.jpg');

    % Convert the image to double precision
    grayWorldImage = im2double(image);

    %% Gray World
    meanR = mean(mean(grayWorldImage(:,:,1)));
    meanG = mean(mean(grayWorldImage(:,:,2)));
    meanB = mean(mean(grayWorldImage(:,:,3)));

    meanGray = (meanR + meanG + meanB) / 3;

grayWorldImage(:,:,1) = grayWorldImage(:,:,1) * 0.5 / meanR;
grayWorldImage(:,:,2) = grayWorldImage(:,:,2) * 0.5 / meanG;
grayWorldImage(:,:,3) = grayWorldImage(:,:,3) * 0.5 / meanB;

    %% White World Assumption
    whiteWorldImage = im2double(image);
    maxR = max(max(whiteWorldImage(:, :, 1)));
    maxG = max(max(whiteWorldImage(:, :, 2)));
    maxB = max(max(whiteWorldImage(:, :, 3)));

    maxWhite = max([maxR, maxG, maxB]);

    whiteWorldImage(:, :, 1) = whiteWorldImage(:, :, 1) * maxWhite / maxR;
    whiteWorldImage(:, :, 2) = whiteWorldImage(:, :, 2) * maxWhite / maxG;
    whiteWorldImage(:, :, 3) = whiteWorldImage(:, :, 3) * maxWhite / maxB;

    %% Display the corrected images
    figure;
    subplot(3, 1, 1);
    imshow(image);
    title('Original figure');
    subplot(3, 1, 2);
    imshow(grayWorldImage);
    title('Gray World Assumption');
    subplot(3, 1, 3);
    imshow(whiteWorldImage);
    title('White World Assumption');
```

```
    %% Load the correct white-balanced image
    correctImage = imread('michelangelo_correct.jpg');
    correctImage = im2double(correctImage);
    % correctImage = double(correctImage);

    % Load the three output images
    outputImage = imread('result.jpg');
    outputImage = im2double(outputImage);
    % outputImage = double(outputImage);

    % Calculate PSNR
    [psnr1, ¬] = psnr(grayWorldImage, correctImage);
    [psnr2, ¬] = psnr(whiteWorldImage, correctImage);
    [psnr3, ¬] = psnr(outputImage, correctImage);

    % Calculate SSIM
    ssim1 = ssim(grayWorldImage, correctImage);
    ssim2 = ssim(whiteWorldImage, correctImage);
    ssim3 = ssim(outputImage, correctImage);

    % Calculate LIP-error using the computeFLIP function
    addpath('flip-matlab_0/matlab')
    err1 = computeFLIP(correctImage, grayWorldImage);
    err2 = computeFLIP(correctImage, whiteWorldImage);
    err3 = computeFLIP(correctImage, outputImage);

    % Display the results
    fprintf('PSNR:\n');
    fprintf('Gray World Image: %.2f\n', psnr1);
    fprintf('White World Image: %.2f\n', psnr2);
    fprintf('WB Image : %.2f\n\n', psnr3);

    fprintf('SSIM:\n');
    fprintf('Gray World Image: %.4f\n', ssim1);
    fprintf('White World Image: %.4f\n', ssim2);
    fprintf('WB Image : %.4f\n\n', ssim3);

    fprintf('LIP-error:\n');
    fprintf('Gray World Image: %.2f\n', err1);
    fprintf('White World Image: %.2f\n', err2);
    fprintf('WB Image : %.2f\n', err3);
    end
```

The result is as follows.

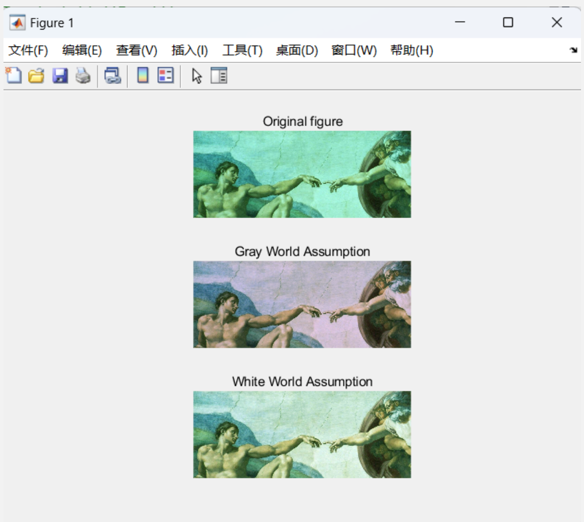


Figure 1: Task1

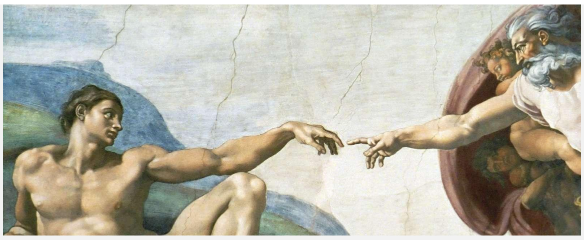


Figure 2: Task1 Ground Truth

The ground truth is as 2. Based on the evaluation metrics and subjective visual comparison, it can be observed that the gray world assumption is closer to the original image. Although there is some color deficiency in the background during the restoration process, it still does a good job of preserving the original image. .

Table 1: Task 1 Result.

	PSNR	**SSIM**	**LIP**
Gray World Image	18.83	0.6853	0.48
White World Image	22.78	0.6316	0.33
WB Image	27.04	0.8882	0.25

Answer to Task 2 Segmentation with Graph Cuts

Firstly, I loaded the cardiac image data and defines the image's dimensions (height and width). Then, estimated the mean and standard deviation of the ventricle and background regions, which will be used in the subsequent segmentation process. Next, I established the topological structure of the image by creating a connectivity matrix between pixels, where pixels are considered as nodes in the graph, and connections are edges. This step is a critical part of the image segmentation process.

Then calculated the image gradient to determine the weights between neighboring pixels, which are used to measure the dissimilarity between different regions. A regularization term is set based on the length of curves, and by adjusting the parameter lambda, control the influence of curve length on the segmentation result.

Subsequently, I assigned data terms to each pixel by calculating the negative log-likelihood of pixel values, indicating the likelihood of each pixel belonging to the ventricle or background. Then, I combined the regularization and data terms using a graphical matrix for the graph-cut algorithm.

Finally, the graph-cut algorithm is run using the maxflow function to find the best segmentation result, dividing the pixels in the image into two regions: ventricle and background. The segmentation mask is then used for visualization, displaying the segmented result of the image. The code is as follows.

```
%% Matlab stub for task 2 in assignment 4 in Image analysis

load heart_data % load data

M = 96; % height of image, change this!
N = 96; % width of image, change this!

n = M*N; % Number of image pixels

%% Estimate the means and the standard
chamber= chamber_values;
background= background_values;

chamber_mean = mean(chamber);
```

```
chamber_std = std(chamber);

background_mean = mean(background);
background_std = std(background);

fprintf('Chamber: mean = %f, std = %f\n', chamber_mean, chamber_std);
fprintf('Background: mean = %f, std = %f\n', background_mean, background_std);

%%
% create neighbour structure

Neighbours = edges8connected(M,N); % use 4-neighbours (or 8-neighbours with ...
    edges8connected)

i=Neighbours(:,1);
j=Neighbours(:,2);
A = sparse(i,j,1,n,n); % create sparse matrix of connections between pixels


% We can make A into a graph, and show it (test this for example for M = 5, N = 6 to
% see. For the full image it's not easy to see structure)
Ag = graph(A);
plot(Ag);

% Choose weights:
[Gx, Gy] = gradient(double(im));
gradient_i = sqrt(Gx(i).^2 + Gy(i).^2);
gradient_j = sqrt(Gx(j).^2 + Gy(j).^2);
weights = abs(gradient_i - gradient_j);

% Decide how important a short curve length is:
lambda = 0.2;


A = sparse(i,j,weights,n,n); % set regularization term so  that A_ij = lambda


mu1 = normfit(background_values);
mu2 = normfit(chamber_values);

pixels = im(:);
neg_log_likelihood_chamber = (pixels - chamber_mean).^2 / (2 * chamber_std^2);
neg_log_likelihood_background = (pixels - background_mean).^2 / (2 * ...
    background_std^2);
Tt = sparse((pixels - chamber_mean).^2 / (2 * chamber_std^2)); % ...
            Ł
Ts = sparse((pixels - background_mean).^2 / (2 * background_std^2)); % ...
            Ł

% create matrix of the full graph, adding source and sink as nodes n+1 and
% n+2 respectively

F = sparse(zeros(n+2,n+2));
F(1:n,1:n) = A; % set regularization weights
F(n+1,1:n) = Ts'; % set data terms
F(1:n,n+1) = Ts; % set data terms
F(n+2,1:n) = Tt'; % set data terms
F(1:n,n+2) = Tt; % set data terms

% make sure that you understand what the matrix F represents!

Fg = graph(F); % turn F into a graph Fg

% help maxflow % see how Matlab's maxflow function works

[MF,GF,CS,CT] = maxflow(Fg,n+1,n+2); % run maxflow on graph with source node (n+1) ...
    and sink node (n+2)

% disp(MF) % shows the optmization value (maybe not so interesting)

```

```
% CS contains the pixels connected to the source node (including the source
% node n+1 as final entry (CT contains the sink nodes).

% We can construct out segmentation mask using these indices
seg = zeros(M,N);
seg(CS(1:end-1)) = 1; % set source pixels to 1
figure;
imagesc(seg); % show segmentation
```

The input image and segment image is as follows.

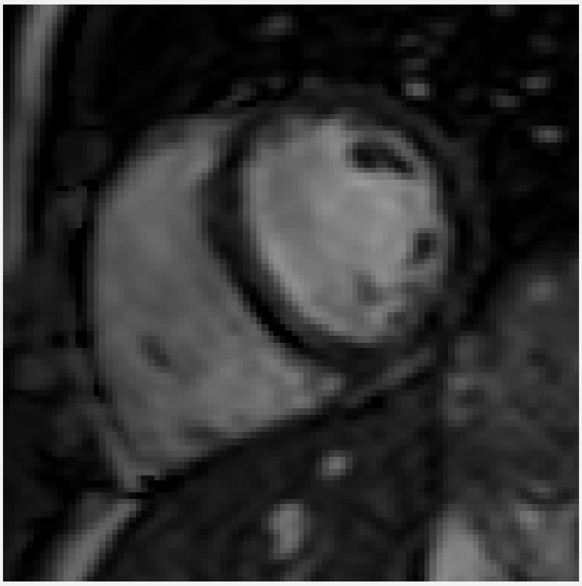


Figure 3: Task 2 Input

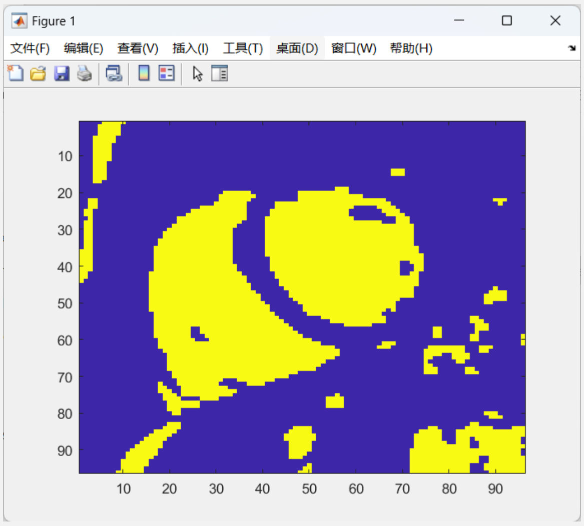


Figure 4: Task 2 Segment

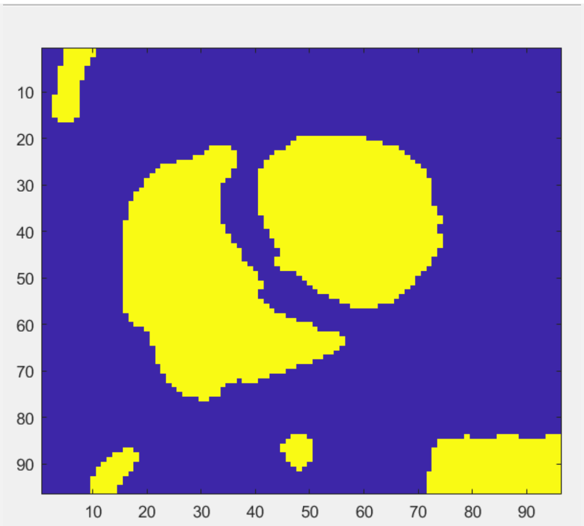


Figure 5: Task 2 Segment Tuned

I made adjustments to the parameters by increasing the values of lambda and weight. After fine-tuning, the image looks as depicted in Figure 5, and it's evident that the unwanted segmented areas have been significantly reduced.

Table 2: Task 2.

	mean	**std**
Chamber	0.354511	0.099156
Background	0.104257	0.096724

Answer to Computer Vision

x_1 is the coordinate of point X in image 1, usually expressed as a homogeneous coordinate,like $x_1 = [u_1, v_1, 1]^T$

x_2 is the coordinate of point X in image 2, usually expressed as a homogeneous coordinate,like $x_1 = [u_1, v_1, 1]^T$

The following equation is satisfied for the Epipolar Constraint:

$$x_2^T \cdot F \cdot x_1 = 0 \tag{1}$$

Substituting the coordinates of all points into the formula, if they satisfy the Epipolar constraint, they correspond to the respective points.

```
clc
clear all
% Define points in homogeneous coordinates
a1 = [-4; 5; 1];
a2 = [3; -7; 1];
a3 = [-10; 5; 1];
b1 = [3; 2; 1];
b2 = [6; -1; 1];
b3 = [2; -2; 1];

% Define the fundamental matrix F
F = [2, 2, 4;
     3, 3, 6;
    -5, -10, -6];
```

```
% Calculate a' * F * b for all combinations
result_b1_a1 = b1' * F * a1;
result_b1_a2 = b1' * F * a2;
result_b1_a3 = b1' * F * a3;
result_b2_a1 = b2' * F * a1;
result_b2_a2 = b2' * F * a2;
result_b2_a3 = b2' * F * a3;
result_b3_a1 = b3' * F * a1;
result_b3_a2 = b3' * F * a2;
result_b3_a3 = b3' * F * a3;
```

The result is as follows.

result_b1_a1	0
result_b1_a2	25
result_b1_a3	-42
result_b2_a1	-9
result_b2_a2	31
result_b2_a3	-33
result_b3_a1	-42
result_b3_a2	53
result_b3_a3	0

Figure 6: Task 3

So, a1 corresponds to b1, and a3 corresponds to b3.

Answer to OCR system construction and system testing

In my previous work, for the feature extraction part , I took into consideration that in the 'home1' test set, the scale may vary. To address this, I utilized scale-independent features. Firstly, Hoff circle detection was employed to identify the number of circles present in each component. Secondly, boundary values within the components were extracted, transformed into bounding boxes, and used as regions of interest (ROIs). These ROIs were then equally divided into 9 parts, and the ratio of white pixels to black pixels in each part was computed.

In the classification part, I opted for SVM as the classifier, as it demonstrated the best performance in task 2.

The overall hitrate of the previous work on the five dataset are shown in Table 3.

The hitrate clearly shows that, the results were not very satisfactory, and I attribute this to the limited number of scale-invariant features. To address this issue, I surrounded the ROIs with black pixels and then resized them into square images of 28x28, thus standardizing the scale. Subsequently, I used a convolutional neural network for training, and the training results are shown in the figure 7. When using the trained model for prediction, the accuracy is as depicted in the figure 8 and table 4.

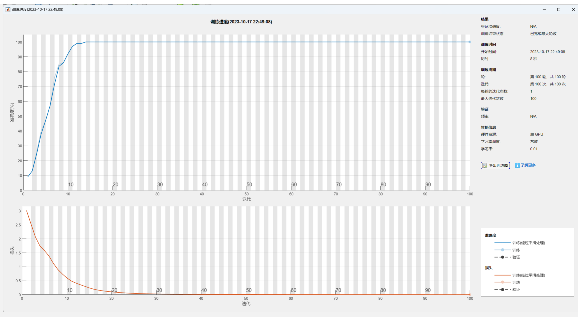


Figure 7: CNN Train

```
>> inl4_test_and_benchmark
Hitrate = 82%
>> inl4_test_and_benchmark
Hitrate = 74%
>> inl4_test_and_benchmark
Hitrate = 73.8%
>> inl4_test_and_benchmark
Hitrate = 74.5%
>> inl4_test_and_benchmark
Hitrate = 57.2%
```

Figure 8: CNN Result

The code is shown as follow. img2segment:

```
function [S] = im2segment(img)
img = uint8(img);
% figure;
% imshow(img);

% Specify the standard deviation of the Gaussian filter (to control the degree of ...
    blurring)
sigma = 0.5;

% Perform Gaussian filtering
%img = imgaussfilt(img, sigma);

 img = imbilatfilt(img);
% img = medfilt2(img, [3, 3])

threshold = 0.158; % Set the threshold of image binarize
binary_image = imbinarize(img, threshold); % binarize
% imshow(binary_image);

%% 8-connected components
labeledImage = bwlabel(binary_image, 8);
minPixels = 1 ; % set min pixel num


%% Extracting information about connected components
stats = regionprops(labeledImage, 'BoundingBox', 'PixelIdxList');

% Initialize an array of cells for storing segmented images
numStats = numel(stats);
S = cell(1, numStats);

% Create an array of flags to keep track of merged cells
```

```
merged = false(1, numStats);
small = false(1, numStats);

%% Storing coordinate indexes of different labels in cells
for i = 1:numStats
    % Get the coordinate index of the current connected component
    pixelIdxList = stats(i).PixelIdxList;

    % Create a segmented image of the same size as the original image
    segmented_image = zeros(size(labeledImage));
    segmented_image(pixelIdxList) = 1;
    numPixels = sum(segmented_image(:) == 1);
    if numPixels < minPixels
        small(i)=true;
    end
    % Storing Segmented Images into Cells
    S{i} = segmented_image;
end


%%
Dis_threshold = 20; % distance threshold

% Calculate the center coordinates of each cell and combine them
for i = 1:numStats
    if ¬merged(i) && ¬small(i)
        for j = i+1:numStats
            if ¬merged(j) && ¬small(j)
                % Calculate the center coordinates
                centro_i = regionprops(S{i}, 'Centroid');
                centro_j = regionprops(S{j}, 'Centroid');

                % Extracting the center coordinates
                centro_i = centro_i.Centroid;
                centro_j = centro_j.Centroid;

                % culculate the Euclidean distance
                distance = norm(centro_i - centro_j);

                if distance < Dis_threshold
                    % Merge two cells and add elements to the first cell
                    S{i} = S{i} | S{j};
                    merged(j) = true;
                end
            end
        end
    end
end

S = S(¬merged);
end
```

segment2features:

```
function features = segment2features(I)
%%
stats = regionprops(I, 'BoundingBox');
bounding_box = stats.BoundingBox;

%      ł  1
x = bounding_box(1);
y = bounding_box(2);
width = ceil(bounding_box(3));
height = ceil(bounding_box(4));

%      ł  2           ł
newWidth = width + 4;  %
newHeight = height + 4;
```

```
newImage = zeros(newHeight, newWidth, 'uint8');  %

%      ł  3
newImage(3:end-2, 3:end-2) = I(y:y+height-1, x:x+width-1);

%     ł  4                 2 8 x 2 8
scaledImage = imresize(newImage, [28, 28]);

flattenedImage = reshape(scaledImage, [],1);

%%

% % %      H O G
 hog_features = extractHOGFeatures(scaledImage);
 hog_features = hog_features';
%%
features = flattenedImage;
```

CNN Training;

```
function net = trainSimpleCNN(X,Y)

% reformat input data
% input images assumed to be 19x19
nx = 28;
ny = 28;
X = reshape(X,ny,nx,1,[]);

% classes should be of type categorical
Y = categorical(Y);

% define cnn-structure
layers = [
    imageInputLayer([ny nx 1])

    convolution2dLayer(3,8,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,16,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,64,'Padding','same')
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];


% specify training options

options = trainingOptions('sgdm', ...
    'InitialLearnRate',0.01, ...
    'MaxEpochs',100, ...
    'Shuffle','every-epoch', ...
```

```
    'Verbose',false, ...
    'Plots','training-progress');

% train network

net = trainNetwork(X,Y,layers,options);
```

CNN Predict:

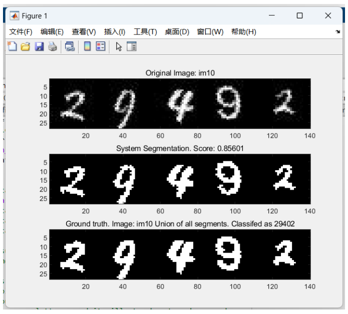
```
function predictedClass = predictSimpleCNN(X,net)

% reformat input data
% input images assumed to be 19x19
nx = 28;
ny = 28;
X = reshape(X,ny,nx,1,[]);

% predict classes
predictions = predict(net,X);

[maxValue, predictedClass] = max(predictions);
end
```

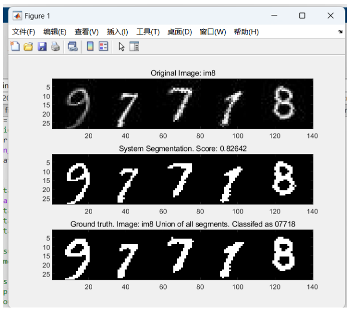
Table 3: Previous Hitrate.

	short1	**short2**	**home1**	**home2**	**home3**
Hitrate	60%	52%	57.6%	59.2%	43.5%

Table 4: CNN Hitrate.

	short1	**short2**	**home1**	**home2**	**home3**
Hitrate	82%	74%	73.8%	74.5%	57.2%


(a) figure 1


(b) figure 2

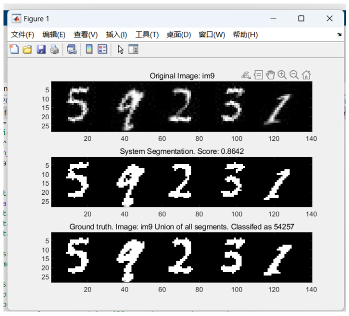

(c) figure 3

Figure 9: classification result