Image Analysis Assignment 4

Ruqayyah Nabage

October 2023

1 Color Correction of Images

- Gray World Assumption: To implement the gray world assumption, I calculated the mean of each color channel and found the scaling factor for each color channel; using the following formula:

$$\begin{pmatrix}0.5\\0.5\\0.5\end{pmatrix} = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N} Af(i,j) \text{ and } A = \begin{pmatrix}\alpha & 0 & 0\\ 0 & \beta & 0\\ 0 & 0 & \gamma\end{pmatrix}$$

$$\alpha = \frac{0.5}{\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N} Af_{red}(i,j)}$$

$$\beta = \frac{0.5}{\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N} Af_{green}(i,j)}$$

$$\gamma = \frac{0.5}{\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N} Af_{blue}(i,j)}$$

  The final image is a concatenation of each channel multiplied by it's respective scaling factor.

- White World Assumption: To implement the white world, I followed the same steps as the gray world algortihm except instead of calculating the mean of each channel, I found the max of each channel and used it to calculate the respective scaling factors.

$$\alpha = \frac{1}{\max_{i,j} f_{red}(i,j)} \quad \beta = \frac{1}{\max_{i,j} f_{green}(i,j)} \quad \gamma = \frac{1}{\max_{i,j} f_{blue}(i,j)}$$

```matlab
im = imread("michelangelo_colorshift.jpg"); % reading image
im = double(im)/255;    %this division is to scale the range of
    colors to be between 0 and 1; just to make things on the same
    scale

%the mean of each color channel
r_mean = mean(im(:,:,1),"all");
```

```
g_mean = mean(im(:,:,2),"all");
b_mean = mean(im(:,:,3),"all");

%scaling factor of each channel
g_alpha = 0.5/r_mean;
g_beta = 0.5/g_mean;
g_gamma = 0.5/b_mean;

% the final image is just a combination (i.e concatenation along
    dim 3) of each scaling factor x its channel
grey_image = cat(3, g_alpha*im(:,:,1), g_beta*im(:,:,2), g_gamma*im
    (:,:,3));
imwrite(grey_image, 'grey_world.jpg')

%now white world
%the max of each color channel
r_max = max(im(:,:,1),[],"all");
g_max = max(im(:,:,2),[],"all");
b_max = max(im(:,:,3),[],"all");

%scaling factor of each channel
w_alpha = 1/r_max;
w_beta = 1/g_max;
w_gamma = 1/b_max;

%transformed white image
white_image = cat(3, w_alpha*im(:,:,1), w_beta*im(:,:,2), w_gamma*
    im(:,:,3));
imwrite(white_image, 'white_image.jpg')
```

Listing 1: Code for the Gray and White World Implementation

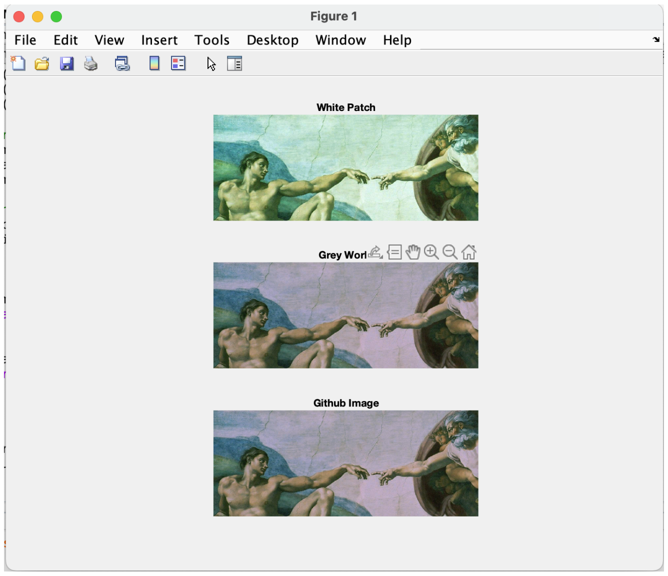

Figure 1: Resulting Output Images

	PSNR	SSIM	Ⅎlip
Gray Image	15.3307	0.9306	0.5932
White Image	32.4291	0.9777	0.3328
Github Image	29.7377	0.9873	0.2451

Table 1: Computed Errors

2 Segmentation with Graph Cuts

```
load heart_data % load data

M = size(im,1); % height of image
N = size(im,2); % width of image

n = M*N; % Number of image pixels

% create neighbour structure
Neighbours = edges8connected(M,N); % used 8-neighbours

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

%% calculating mean 1 and mean 2 of heart data & other statistics
mu1 = mean(chamber_values);
mu2 = mean(background_values);

sd1 = std(chamber_values);
sd2 = std(background_values);

pdf1 = normpdf(im(:),mu1,sd1);
pdf2 = normpdf(im(:),mu2,sd2);

likelihood1 = normlike([mu1,sd1],chamber_values);
likelihood2 = normlike([mu1,sd1],background_values);

% Choose weights:

% Decide how important a short curve length is:
lambda = 1.5;

A = A*lambda; % set regularization term so  that A_ij = lambda

%Ts = -log(pdf1) + 10000000; "finetuning " - attempt at no. 3
Ts = -log(pdf1) - min(-log(pdf1), [],"all"); % set weights to
    source, according to assignment!

%Tt = -log(pdf2) + 10000000;
Tt = -log(pdf2) - min(-log(pdf2), [],"all"); % set weights to sink,
     according to assignment!

% 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

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)

% 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 our segmentation mask using these indices
seg = zeros(M,N);
seg(CS(1:end-1)) = 1; % set source pixels to 1


subplot(1,2,1)
image(im);
title("Input Image")

subplot(1,2,2)
imagesc(seg); % show segmentation
colormap("gray")
title("Segmented Image")
```
Listing 2: Code for the Segmentation with Graph Cuts

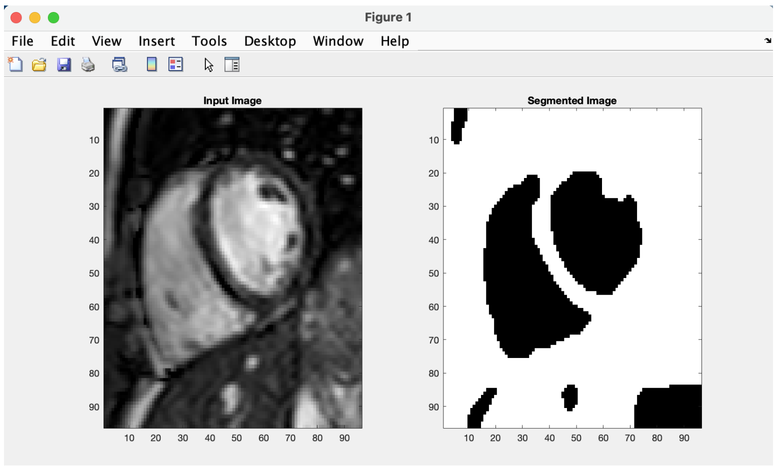

Figure 2: Input Image and Result after Segmentation

3 Computer Vision

Two points in different images are in correspondence if they satisfy the following relation:

$$x_2 F x_1 = 0$$

$$\text{and } x_1 = \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix}, x_2 = \begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix}$$

To find the points in correspondence; each point in Image 1 needs to be tested with every point in Image 2.

$$a_1 = \begin{pmatrix} -4 \\ 5 \\ 1 \end{pmatrix}, \quad a_2 = \begin{pmatrix} 3 \\ -7 \\ 1 \end{pmatrix}, \quad a_3 = \begin{pmatrix} -10 \\ 5 \\ 1 \end{pmatrix}, \quad b_1 = \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}, \quad b_2 = \begin{pmatrix} 6 \\ -1 \\ 1 \end{pmatrix}, \quad b_3 = \begin{pmatrix} 2 \\ -2 \\ 1 \end{pmatrix}$$

$$F = \begin{pmatrix} 2 & 2 & 4 \\ 3 & 3 & 6 \\ -5 & -10 & -6 \end{pmatrix}$$

Checking for points that correspond with a_1

- a_1 and $b_1 \rightarrow b_1^T F a_1 = \begin{pmatrix} 3 & 2 & 1 \end{pmatrix} \times \begin{pmatrix} 2 & 2 & 4 \\ 3 & 3 & 6 \\ -5 & -10 & -6 \end{pmatrix} \times \begin{pmatrix} -4 \\ 5 \\ 1 \end{pmatrix} = 0$

- a_1 and $b_2 \rightarrow b_2^T F a_1 = \begin{pmatrix} 6 & -1 & 1 \end{pmatrix} \times \begin{pmatrix} 2 & 2 & 4 \\ 3 & 3 & 6 \\ -5 & -10 & -6 \end{pmatrix} \times \begin{pmatrix} -4 \\ 5 \\ 1 \end{pmatrix} = -9$

- a_1 and $b_3 \rightarrow b_3^T F a_1 = \begin{pmatrix} 2 & -2 & 1 \end{pmatrix} \times \begin{pmatrix} 2 & 2 & 4 \\ 3 & 3 & 6 \\ -5 & -10 & -6 \end{pmatrix} \times \begin{pmatrix} -4 \\ 5 \\ 1 \end{pmatrix} = -42$

Checking for points that correspond with a_2

- a_2 and $b_1 \rightarrow b_1^T F a_2 = \begin{pmatrix} 3 & 2 & 1 \end{pmatrix} \times \begin{pmatrix} 2 & 2 & 4 \\ 3 & 3 & 6 \\ -5 & -10 & -6 \end{pmatrix} \times \begin{pmatrix} 3 \\ -7 \\ 1 \end{pmatrix} = 25$

- a_2 and $b_2 \rightarrow b_2^T F a_2 = \begin{pmatrix} 6 & -1 & 1 \end{pmatrix} \times \begin{pmatrix} 2 & 2 & 4 \\ 3 & 3 & 6 \\ -5 & -10 & -6 \end{pmatrix} \times \begin{pmatrix} 3 \\ -7 \\ 1 \end{pmatrix} = 31$

- a_2 and $b_3 \rightarrow b_3^T F a_2 = \begin{pmatrix} 2 & -2 & 1 \end{pmatrix} \times \begin{pmatrix} 2 & 2 & 4 \\ 3 & 3 & 6 \\ -5 & -10 & -6 \end{pmatrix} \times \begin{pmatrix} 3 \\ -7 \\ 1 \end{pmatrix} = 53$

Checking for points that correspond with a_3

- a_3 and $b_1 \rightarrow \ b_1^T F a_3 = \begin{pmatrix} 3 & 2 & 1 \end{pmatrix} \times \begin{pmatrix} 2 & 2 & 4 \\ 3 & 3 & 6 \\ -5 & -10 & -6 \end{pmatrix} \times \begin{pmatrix} -10 \\ 5 \\ 1 \end{pmatrix} = -42$
- a_3 and $b_2 \rightarrow \ b_2^T F a_3 = \begin{pmatrix} 6 & -1 & 1 \end{pmatrix} \times \begin{pmatrix} 2 & 2 & 4 \\ 3 & 3 & 6 \\ -5 & -10 & -6 \end{pmatrix} \times \begin{pmatrix} -10 \\ 5 \\ 1 \end{pmatrix} = -33$
- a_3 and $b_3 \rightarrow \ b_3^T F a_3 = \begin{pmatrix} 2 & -2 & 1 \end{pmatrix} \times \begin{pmatrix} 2 & 2 & 4 \\ 3 & 3 & 6 \\ -5 & -10 & -6 \end{pmatrix} \times \begin{pmatrix} -10 \\ 5 \\ 1 \end{pmatrix} = 0$

From these results, the only points that correspond with each other are a_1 & b_1 and a_3 & b_3. This is because two points can be projections of the same 3D point if they satisfy the relation $x_2 F x_1 = 0$ where F is the fundamental matrix and x_1 and x_2 are the points expressed in homogeneous coordinates. That is why I tested the combinations above; and from the combinations only a_1 & b_1 and a_3 & b_3 gave a result of 0.

4 OCR system construction and system testing

Dataset	Hitrate
short1	46%
short2	50%
home1	9.9%
home2	9.8%
home3	9.8%

Table 2: Hitrates for System Version 1

My system version 1 had good results for the short1 and short2 datasets because after choosing my features in assignment 2, I spent even more time on my features and normalizing them in Assignment 3. The normalization I had done was by manually checking the output values from each feature from each number; finding the minimum and maximum values and performing min max scaling on each feature. For example; a feature I had was the length of the longest vertical line in a number and I had normalized it as follows: vertical = (vertical - 5)/(19-5); This works well enough for the images in short1 and short2 because they generally have the same size of numbers and are fairly similar to each other. But also, that is why it works terribly on the images in the home dataset; the numbers in that dataset have entirely different sizes.

Because of that, the first adjustment I made was to change the way I had performed the normalization of my features. I decided to use the properties gotten from the area the digit was in, in the image to normalize most of the features. I got this using the regionprops Bounding Box property. I got the starting x, y coordinates, the width and height of the bounding box. The x coordinate corresponded to the starting column of the digit and the y coordinate corresponded to the starting row of the image. The width of the bounding box plus the starting column gave me the ending column of the digit. The height of the bounding box plus the starting row gave me the ending row of the digit. The features I had at the time were:

- The horizontal edges of the image (gotten as a result of a convolution of the image with a G_x Sobel filter and a histogram of it taken in 4 bins with the 0 bin discarded). I changed it to be normalized by the dividing by the sum of all the bins.
- The vertical edges of the image; the same procedure as above except using a G_y Sobel filter instead.
- The perimeter of the digit using bwperimeter; I normalized this by dividing by the perimeter of the bounding box which I obtained by calculating $2 \times (width + height)$.
- The longest vertical line in the image; I normalized this by dividing by the height of the bounding box.
- The longest horizontal line in the image; I normalized this by dividing by the width of the image.
- The position of the longest vertical line; I normalized this by performing a pseudo-min max scaling. The lowest/first possible position it could be is the starting column of the bounding box and the highest/last possible position it could be is the last column of the bounding box which I mentioned how I obtained above. I then normalized the position as; $\frac{vertical\_position - starting\_column}{width}$.
- The position of the longest horizontal line; I normalized the same way as the longest vertical line except using row properties and height.
- The number of holes in an image; this I had normalized previously as $\frac{conn-1}{3-1}$ perfoming min max as mentioned earlier, I just left it as it was without normalizing since the highest possible value it could give for any digit is 3 regardless of the shape.

After, I effected these changes the performance of my system went down to $\tilde{2}8\%$ on the short datasets and went up to $\tilde{3}3\%$ on the home datasets. I tried tweaking my features and the normalization but the performance did not go up. I decided to change my classifier as I had been using my nearest neighbour classifier that I had written in Assignment 3, task 1. I used a multiple SVM classifer with the fitcecoc function. It didn't make much of a difference. I used the mode 2 debugging system and I noticed that particularly with the short datasets some of my segments were cut up into segments so I decided to improve on my segmentation (im2segment).

I added a median filter as home3 dataset in particular had a lot of salt and pepper noise. Next, I tried various methods; first: I counted the number of segments gotten from the bwlabel (which I used for my segmentation) and working on the assumption that all the images to be segmented would have 5 numbers, I had bwlabel output the number of connected components as well. If the image had more than 5 connected components, then bwmorph(”majority”) operation would be carried out on the image to try and close the holes in the image. Then bwlabel would be done again and the number of connected components counted again. I put this in a while loop and as long as more than 5 connected components existed it would keep trying to close the gap. This made the system get stuck in infinite loop on some images in which it wasn’t able to close the faint lines completely.

Next; I tried a similar tactic but using bwmorph(”thicken”) operation a few times and I created an additional loop for the images that had had more than 5 segments initially before the thicken operation; a bwmorph(”thin”) operation was carried out to try and restore the number to its original shape. But this also yielded slightly deformed numbers.

The method which worked was removing the initial closing (imclose) I had been perfoming on the image after thresholding and replacing it with a bwareaopen to fill in the holes. I made this replacement because even though imclose had worked, it sometimes closed the holes of some numbers like 8 and 9 which were details I needed for the number of holes feature I had chosen. I also reduced my threshold value to pick up on fainter parts of the image. To deal with the noise this low threshold also picked up on I added a conditional also working on the assumption that there would only be 5 digits in the image. If there were more than 5 connected components, their areas were found by using regionprops on the connected components and any components with areas less than 25 (I set this threshold through experimenting with the different datasets) were discarded, and the new image with the discarded parts was segmented again using bwlabel. This is the final im2sement.

```matlab
function S = im2segment(im)

im = uint8(im); %converting the image into 8 bit

% im_f = imgaussfilt(im); %filtering the image for noise
im_f = medfilt2(im,[2 2]);

bnd = 28; % set some arbitrary bound on thresholding
im_t = im_f>bnd; % threshold the image
im_d = bwareaopen(im_t,3);

L = bwlabel(im_d); % label connected components in image
nrseg = max(L(:)); % find out number of segments

% discarding any segments with area less than 25%
if nrseg > 5
    connComp = bwconncomp(im_d);
    stats = regionprops(connComp, 'Area');
    thresholdArea = 25; % set by experimenting
    for i = 1:length(stats)
        if stats(i).Area < thresholdArea
            im_d(connComp.PixelIdxList{i}) = 0;
        end
    end
    L = bwlabel(im_d);
    nrseg = max(L(:));
end

S = cell(1,nrseg); %creating the cell array S

for kk = 1:nrseg    %created a for loop to store the different
    segments in
        S{kk} = (L==kk);
end
```

Listing 3: Final im2segment code

These changes made the short datasets performance go up to $\tilde{3}3\%$ but the home datasets performance didn't change. I decided to add new features to my segment2features. Some of the features I tried were:

- The ratio of white pixels in a digit to the total number of pixels in the bounding box.
- The ratio of white pixels in the top half of the bounding box to the total number of pixels in the top half of the image.
- The same as above except with the bottom, the right and left side of the image. What I was trying to achieve with these features was trying to find the symmetry and distribution of pixels in a particular digit.
- A feature that was the comparison of the number of pixels in the top half of the image and the bottom half of the digit.
- And one that was a comparison of the left and the right parts of the digit.
- Instead of looking for the longest vertical line in the digit and its position, I decided to look at the number of pixels vertically at certain specific positions in each digit (the center, the quarter and three quarters of the bounding box). I did the same for the horizontal aspect as well.
- In my initial segment2features in Assignment 2, I had used some regionprops features as well but I had applied them on the entire image instead of on the area marked by the bounding box alone. I tried using different combinations of the properties. The ones that I found the most useful were the Eccentricity, Extent, Euler Number, and Solidity.
- As the Extent is the ratio of pixels in the region to pixels in the total bounding box which was more or less the same as the ratio feature I had developed earlier, I removed the ratio feature I had been computing manually.

- Since the eccentricity, Euler number and solidity had worked well and they were all in a way describing the roundness/ round properties of the digits; I decided to add Compactness as a feature as well; which is a measure of how closely the shape's area or perimeter approaches that of a perfect circle. This is calculated by $\frac{Perimeter^2}{4 \times \pi \times Area}$.

I tried several different combinations of my features; the old ones that I mentioned initially and the new ones mentioned above. I realized in the combinations where I had the horizontal and vertical edges the hitrates went down so I took them out. The ratios of the top, bottom, right and left didn't seem very useful as well; the comparisons made more impact. In the end my final features were:

- Number of holes (conn)
- If the left side had more pixels than the right (com_left)
- If the right side had more pixels than the left (com_right)
- If the top side had more pixels than the bottom (com_top)
- If the bottom side had more pixels than the top (com_bottom)
- Number of pixels in the middle column of the bounding box (vertical_center)
- Number of pixels at the $\frac{1}{4}$th column of the bounding box (vertical_quarter)
- Number of pixels at the $\frac{3}{4}$th column of the bounding box (vertical_3quarter)
- Number of pixels in the middle row of the bounding box (horizontal_center)
- Number of pixels at the $\frac{1}{4}$th row of the bounding box (horizontal_quarter)
- Number of pixels at the $\frac{3}{4}$th row of the bounding box (horizontal_3quarter)
- The ratio of the width of the bounding box to the height (ratio)
- The Euler number (enum), eccentricity, extent, solidity and compactness

This is the final version of my segments2features

```
function features = segment2features(If)

%first step; filter the image to remove noise; I'll use a median
    filter

%second step; centering the image
% Find centroid.
prop = regionprops(If, 'Centroid');
```

```matlab
% Translate the image.
centepide = prop.Centroid;
xt = centepide(1,1);
yt = centepide(1,2);
% Get center of image
[rows, columns] = size(If);
xc = columns/2;
yc = rows/2;
deltax = xc - xt;
deltay = yc - yt;
Ic = imtranslate(If,[deltax deltay],'FillValues', 0);


% you will be the backbone of my normalizations
props = regionprops(Ic,'BoundingBox');
box = props.BoundingBox;
x_start = round(box(1,1)); %starting column
y_start = round(box(1,2)); %starting row
if y_start > 1
    y_start = y_start-1;
end

width = box(1,3); %will be  used to normalize longest line
height = box(1,4);

%ratio of width to height
ratio = width/height;

x_end = round(x_start + width)-1;
y_end = round(y_start + height)-1;

ncol = round(width); %columns in bounding box
nrow = round(height); %rows in bounding box

%image size
row = size(Ic,1);
column = size(Ic,2);

% top to bottom half of the image
half = round(row/2);
if (row/2 ~= half)
    half = half-1;
end


%half of bounding box
col_half = round(ncol/2);

if col_half == 0
    col_half = 1;
elseif (ncol/2 ~= col_half)
    col_half = col_half-1;
end

row_half = round(nrow/2);

if row_half ==0
```

```
    row_half = 1;
elseif (nrow/2 ~= row_half)
    row_half = row_half-1;
end

% dividing the image-bounding box
bounding_box = Ic(y_start:y_end,x_start:x_end);
top = Ic(y_start:y_start+row_half,x_start:x_end);
bottom = Ic(y_start+row_half:y_end,x_start:x_end);
left = Ic(y_start:y_end,x_start:x_start+col_half);
right = Ic(y_start:y_end,x_start+col_half:x_end);

% ratio of white pixels to total number of pixels in top of
    bounding box
nrnnz_top = nnz(top);
pixels_topbot = ncol*nrow/2;
ratio_top = (nrnnz_top/pixels_topbot);

% ratio of white pixels to total number of pixels in bottom of
    bounding box
nrnnz_bottom = nnz(bottom);
ratio_bottom = (nrnnz_bottom/pixels_topbot);

% ratio of white pixels to total number of pixels in left of
    bounding box
nrnnz_left = nnz(left);
pixels_side = ncol/2*nrow;
ratio_left = (nrnnz_left/pixels_side);

% ratio of white pixels to total number of pixels in right of
    bounding box
nrnnz_right = nnz(right);
ratio_right = (nrnnz_right/pixels_side);

com_top = 0;
com_bottom = 0;
com_left = 0;
com_right = 0;

%if top has more pixels than bottom
if nrnnz_top > nrnnz_bottom
    com_top = 1;
elseif nrnnz_top < nrnnz_bottom
    com_bottom = 1;
elseif nrnnz_top == nrnnz_bottom
    com_top = 1;
    com_bottom = 1;
end

%if left or right has more pixels
if nrnnz_right > nrnnz_left
    com_right = 1;
elseif nrnnz_right < nrnnz_left
    com_left = 1;
elseif nrnnz_right == nrnnz_left
    com_right =1;
    com_left = 1;
```

```
end

vertical_center = sum(bounding_box(:, col_half))/height;
vertical_quarter = sum(bounding_box(:, round(col_half/2)))/height;
vertical_3quarter = sum(bounding_box(:, round(col_half/4*3)))/
    height;
horizontal_center = sum(bounding_box(row_half, :))/width;
horizontal_quarter = sum(bounding_box(round(row_half/2), :))/width;
horizontal_3quarter = sum(bounding_box(round(nrow/4*3), :))/width;

% regionprops features
stats = regionprops(bounding_box, 'Eccentricity', 'Extent', '
    EulerNumber', 'Solidity', 'Area', 'Perimeter');
perimeter = stats.Perimeter;
area = stats.Area;
eccentricity = stats.Eccentricity;
extent = stats.Extent;
enum = stats.EulerNumber;
solidity = stats.Solidity;
compactness = ((perimeter .^ 2) ./ (4 * pi * area))/5;

% no. of holes in image
inverse = ~Ic;
[~ ,conn] = bwlabel(inverse);

features = [conn com_left com_right com_bottom com_top
    vertical_center vertical_quarter vertical_3quarter
    horizontal_center horizontal_quarter horizontal_3quarter ratio
    eccentricity extent enum solidity compactness];
```

Listing 4: Final segments2features code

The final change I effected was to change my classifier to a random forest classifier, as it generalizes better on large datasets. This is the final version of my features2class and class_train.

```

function y = features2class(x, classification_data)

ycell = predict(classification_data,x.');
y = str2double(cell2mat(ycell));

end
```

Listing 5: Final features2class code

```

function classification_data = class_train(X, Y)

classification_data = TreeBagger(50,X.',Y.');

end
```

Listing 6: Final class_train code

My hitrates for my system version 2 are as follows:

Dataset	Hitrate
short1	50%
short2	44%
home1	55.7%
home2	52.1%
home3	51.4%

Table 3: Hitrates for System Version 2