Assignment 4 - Image Analysis

Theodora Gaiceanu

1 Color correction of images

1.1

Saturation is responsible for the way the colours look (if they are vivid or if they are very pale). Consequently, in this problem we want to make an under-saturated image look more vivid. In order to do so, one can use the *rgb2hsv* Matlab built-in function in order to convert the RGB colour map to an HSV (hue-saturation-value) colour map. After that, saturation can be accessed, as it is the second column of the matrix returned by *rgb2hsv* function. Saturation can be enhanced using histogram equalisation upon the saturation channel. In this way, the range of saturation values will expand, the effect being a more vivid look for the colours. This can be done using the *histeq* built-in Matlab function. After the multiplication is done, the image must be converted back to RGB.
The code for this subsection can be noticed below.

```
%%1. make the colors more vivid
img_1 = imread('arcimboldo_low.jpg');
figure;
imagesc(img_1);
title("Original image - 1 ");

%convert the RGB image to HSV
hsv = rgb2hsv(img_1);
%perform histogram equalization for the saturation channel
hsv(:,:,2) = histeq(hsv(:,:,2));
%convert the image back to RGB
img_1_new = hsv2rgb(hsv);

figure;
```

```
subplot(1,2,1), histogram(hsv_init);
title("Original image - histogram");
subplot(1,2,2), histogram(hsv(:,:,2));
title("Modified image - histogram");

figure;
subplot(1,2,1), imagesc(img_1);
title("Original image");
subplot(1,2,2), imagesc(img_1_new);
title("Modified image");
```

Listing 1: Colour correction code

The results of the programme can be observed below.

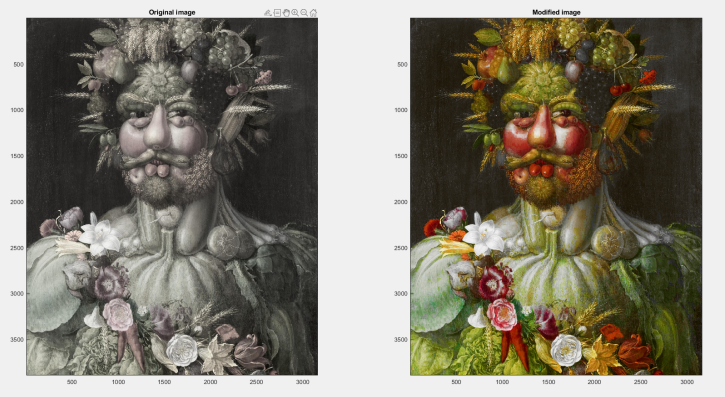

Figure 1: Colour correction

Also, it may be useful to analyse the histograms of saturation channel for the initial image and for the final image. As it can be seen in Figure 2, the saturation for the initial image takes values only in the range 0-0.3. After performing the histogram equalisation, the range of values is 0-1.

1.2

One way to automatically white-balance the image is to equalise all three colour channels of the image. In order to do so, each channel is multiplied with the mean intensity of the gray image and divided with the mean intensity of the channel. In order to get the gray image, *rgb2gray* built-in function

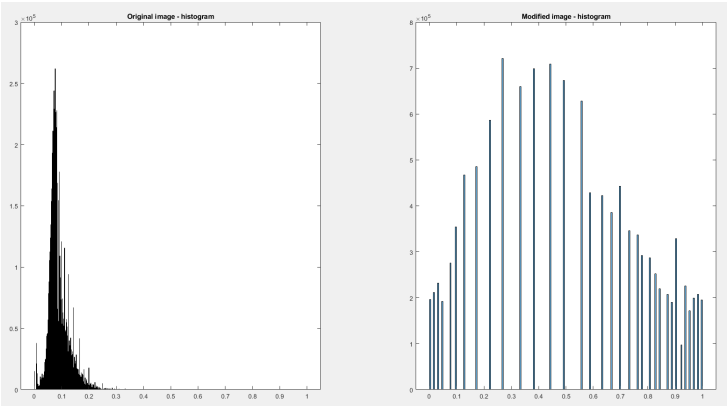

Figure 2: Histogram of the saturation channel for the initial and final image

can be used. After doing the equalisation, the new three colour channels are concatenated.

The code for this subsection can be noticed below.

```
grayImage = rgb2gray(img_2); % convert from RGB to gray image
mean_red = mean2(img_2(:, :, 1)); %red channel
mean_green = mean2(img_2(:, :, 2)); %green channel
mean_blue = mean2(img_2(:, :, 3)); %blue channel
mean_gray = mean2(grayImage);
%equalise the mean for all channels
red = uint8(double(img_2(:, :, 1)) * mean_gray / mean_red);
green = uint8(double(img_2(:, :, 2)) * mean_gray / mean_green
    );
blue = uint8(double(img_2(:, :, 3)) * mean_gray / mean_blue);
%balanced image
img2_new = cat(3, red, green, blue);


figure;
subplot(2,1,1), imagesc(img_2);
title("Original image");
subplot(2,1,2), imagesc(img2_new);
title("Modified image");
```

Listing 2: Code for the white-balanced image

The results can be observed below.

Figure 3: White-balanced image

2 Segmentation with Graph Cuts

The image segmentation using graph cuts is in fact an optimisation problem. There are two regions: the foreground (the chambers) having a constant gray level μ_1, and the background, having a constant gray level μ_0. The goal is to find best curve γ that has the foreground as the inside region Γ and the background as the exterior. But the segmentation problem can be seen in fact as a classification one, where we have two classes (the background and the foreground). Therefore, using statistical interpretation, one can express the Mumford-Shah functional as follows [1]:

$$E_0(\gamma) = \int_{R_1} -log(P(f(x,y)|class1)\,dxdy + \int_{R_2} -log(P(f(x,y)|class2)\,dxdy + \nu|\Gamma| \tag{1}$$

Under the assumption of a Gaussian distribution for both classes, the negative log likelihoods for a pixel of intensity f_i are:

$$P(f_i|chamber) = \frac{1}{2}log(2\pi * std^2_{chamber}) + \frac{1}{2}\frac{(f_i - mean_{chamber})^2}{std^2_{chamber}} \tag{2}$$

$$P(f_i|background) = \frac{1}{2}log(2\pi * std^2_{background}) + \frac{1}{2}\frac{(f_i - mean_{background})^2}{std^2_{background}} \tag{3}$$

	mean	standard deviation
chamber	0.3542	0.0992
background	0.1065	0.0936

Table 1: Values for mean and standard deviation

After obtaining the negative log likelihoods, the optimisation problem can be solved using the maxflow/ min-cut algorithm.

2.1

The estimation of the means and the standard deviations for the chamber class and the background class can be computed very easy, using the *mean* and *std* Matlab built-in functions for the ground-truth values.

The values can be observed in Table 1.

2.2

By replacing the values from 1 in Equations (2) and (3), the following negative log likelihoods can be obtained for the two classes:

$$P(f_i|chamber) = \frac{1}{2}log(2\pi * 0.0992^2) + \frac{1}{2}\frac{(f_i - 0.3542)^2}{0.0992^2} \tag{4}$$

$$P(f_i|background) = \frac{1}{2}log(2\pi * 0.0936^2) + \frac{1}{2}\frac{(f_i - 0.1065)^2}{0.0936^2} \tag{5}$$

I also displayed the negative log likelihoods for the image. By analysing Figure 4, it can be observed that the background pixels are black for the negative log likelihood of the background. Similarly, the chamber pixels are black for the negative likelihood of the foreground.

In the end, the segmented image is obtained. The result can be observed in Figure 5. Here, the white pixels correspond to the chamber and the black

Figure 4: Negative log likelihoods

pixels represent the background. I used 7 for lambda.

Figure 5: Result of the segmentation

The code for this section consists of a script and a function and it can be observed below.

```
clear all;
close all;
clc;
load heart_data.mat

%input image
I = im;
```

```
8
9  %%2.1
10 %compute mean and standard deviation for each class
11 mean_chamber_cls = mean(chamber_values)
12 std_chamber_cls = std(chamber_values)
13 mean_background_cls = mean(background_values)
14 std_background_cls = std(background_values)
15
16 %%2.2
17 %initialise mu0, mu1 and put a proper value for lambda
18 mu0 = mean_background_cls;
19 std0 = std_background_cls;
20 mu1 = mean_chamber_cls;
21 std1 = std_chamber_cls;
22 lambda = 7;
23
24 %function that does the segmentation using graph-cut
25 Theta = segment_image(I,mu0,mu1,std1,std0,lambda);
26 figure;
27 imshow(Theta);
```

Listing 3: Script for task 2

```
1  function Theta = segment_image (I,mu0,mu1,std1,std0,lambda )
2      [M,N] = size(I);
3      img = I;
4      n = M*N; % Number of image pixels
5
6      %get the neighbours and make the adjacency matrix A
7      Neighbors = edges4connected(M,N);
8      i= Neighbors(: ,1);
9      j= Neighbors(: ,2);
10     A = lambda*sparse(i,j,1,n,n);
11
12     %compute the negative log likelihoods
13     neg_log_background = 0.5*log(2*pi*std0^2) + 0.5*(((img -
       mu0).^2)/(std0^2));
14     neg_log_chamber = 0.5*log(2*pi*std1^2) + 0.5*(((img - mu1
       ).^2)/(std1^2));
15
16     %T = [ (img(:) - mu1 ).^2 (img(:) - mu0 ).^2];
17     %data term
18     T = [neg_log_chamber(:), neg_log_background(:)];
19     %T = [ (img(:) - mu1 ).^2 (img(:) - mu0 ).^2];
20     T = sparse(T);
21
```

```
    %solve using min-cut algorithm
    [E,Theta] = maxflow(A,T);
    Theta = reshape(Theta,M,N);
    Theta = double(Theta);
end
```

Listing 4: Function for task 2

3 Computer Vision

It is known that if x and $\overline{x}$ are projections of X in $P_1 = [I, 0]$ and $P_2 = [A, t]$ then

$$\overline{x}Fx = 0, \tag{6}$$

where $F = [t] \times A$ [1].

As, we have P_2, we can identify A as:

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 0 \\ 1 & 2 & 2 \end{pmatrix}$$

and t as:

$$\begin{pmatrix} 3 \\ -2 \\ 0 \end{pmatrix}$$

But what we actually need to do in order to check which points can be in correspondence is to compute the product $b_i^T F a_i$. If the product is 0, then the points are in correspondence. Otherwise, they are not. Before computing the product we need to add a 1 for each a_i and b_i (for the third dimension).

As there are 9 possible combinations of products of the form:

$$b_i^T F a_i, \tag{7}$$

it can be easier to use Matlab for the calculations. Therefore, we initialise in a script F, a_i, b_i. Then, we create two matrices, a and b that have the

columns a_i, respectively b_i. Next, we compute the 9 products in a for loop and check where the result is 0.

The results of the products are the following:

```
b(1) * a(1) =14
b(1) * a(2) =31
b(1) * a(3) =132
b(2) * a(1) =14
b(2) * a(2) =0
b(2) * a(3) =8
b(3) * a(1) =14
b(3) * a(2) =-2
b(3) * a(3) =0
```

Listing 5: Results for task 3

Therefore, by analysing the results, it can be noticed that only the pairs b_2, a_2 and b_3, a_3 can be in correspondence, as they are the only one having the product $b_i^T F a_i$ equal to 0.

The code can be noticed below:

```
clear all;
close all;
clc;

F = [2 2 4; 3 3 6; -5 -10 -6];
a1 = [0; -2; 1];
a2 = [-3; 0; 1];
a3 = [-2; -4; 1];

b1 = [-8; -2; 1];
b2 = [0; 3; 1];
b3 = [4; 1; 1];

a = [a1, a2, a3];
b = [b1, b2, b3];

for i =1:3
    for j=1:3
        res = b(:,i)'*F*a(:,j);
        disp(strcat('b( ',string(i),') * a(',string(j),') = '
   ,string(res)))
```

```
    end
end
```

Listing 6: Script for task 3

4 OCR system construction and system testing

The overall hitrate for version 1 of the system can be seen below:

```
>> inl4_test_and_benchmark
Hitrate = 50%
>> inl4_test_and_benchmark
Hitrate = 30%
>> inl4_test_and_benchmark
Hitrate = 23.7%
>> inl4_test_and_benchmark
Hitrate = 24.8%
>> inl4_test_and_benchmark
Hitrate = 22.6%
```

Listing 7: Results for version 1

As it can be noticed, the hitrate for the last three directories is significantly lower than the one obtained for the short1 directory. This is explainable, because the examples from these directories have more noise than the ones from short1 directory. So the segmentation part should be further improved. Also, not all the features that I use for version 1 are helpful for classification.

For version 2, the overall hitrate of the system can be observed in Listing 8.

```
>> inl4_test_and_benchmark
Hitrate = 54%
>> inl4_test_and_benchmark
Hitrate = 54%
>> inl4_test_and_benchmark
Hitrate = 60.9%
>> inl4_test_and_benchmark
Hitrate = 61.6%
```

```
>> inl4_test_and_benchmark
Hitrate = 59.5%
```

Listing 8: Results for version 2

The performances are better in version 2 because I have made several changes in my code in order to reduce the noise in the data set. And I also improved my features.

4.1 Segmentation

First of all, I improved my segmentation code. Previously, I had problems with digits that were cut during the segmentation. So my goal was to reduce that downside. I filtered the image with a Gaussian filter in order to reduce the noise. Then, I binarized the image. I used the value 43 as a threshold, as it gave me the best results. Next, I filled the isolated interior pixels using the *bwmorph* built-in function. After that, in order to get the entire digit and not separate pieces of it, I did a dilation. I also filtered the image from other potential noise, keeping only the 5 objects with the largest perimeter. Then, I did an erosion, as I didn't want to change the initial dimensions of the digit. After the erosion, I filled again the isolated interior pixels. After the morphological operations, I used the *bwlabel* buitl-in function in order to label and to obtain the numbers of 8 connected components in the image. Finally, I constructed the images for each digit.

4.2 Features

In addition, I also improved my features. I noticed that the digits are not centred and have different sizes. So I extracted only the part of the image having the same size as the bounding box of the digit. Then, I resized the image to the standard size, which is 20x20. Next, I defined my features. As a note, I used only the features that improved the performance of the system. In the following part, I'll describe the used features.

For most of the features, some properties of the image regions given by the Matlab function *regionprops* have been taken into account. Firstly, the digits have different sizes and different number of white pixels. Therefore, it

seemed a good idea to include the the perimeter in the feature vector. I also normalised the perimeter, by dividing it with the numbers of rows multiplied with the numbers of columns of the image.

The centroid may also be useful, as it is the average of the pixels forming a shape. So, depending on the shape of a digit, the centroid has different coordinates. Also, I normalised the Centroid, by dividing the centroid components by the number of columns, respectively the number of rows.

Moreover, some digits are round, while others are not. So, another way to differentiate between them would be a feature that describes this roundness. Therefore, Circularity is a good idea, as it is actually the roundness of objects (the circularity value is computed as $\frac{4*Area*\pi}{Perimeter^2}$) [2]. If the object is a perfect circle, the circularity value is 1 (so digits like 0, 3, 8 are expected to have circularity values closer to 1).

Also, some digits have holes inside (e.g. 0, 6, 8, 9). In this context, the Euler number might be a good property, as it is defined as the difference between the total number of objects in the image and the total number of holes in the objects [3]. Therefore, negative and 0 values of the Euler Number should be expected for the digits with holes.

Moreover, I thought that adding the *Eccentricity* to the feature vector may be a good idea, as it could be very helpful for differentiating between line segments (eccentricity=1) and circle segments (eccentricity 0). The eccentricity is defined as the ratio of the distance between the focus points of the ellipse and its major axis length [2]

Furthermore, the hand-written digits have different orientations. Consequently, I added the Orientation property to the feature vector. This property is basically the value of the angle between the x-axis and the major axis of the ellipse that has the same second-moments as the digit [2]. The range of the value is between -90 and 90, so I normalised it by dividing it with 180.

In addition, the Matlab function *imdilate* does a dilation operation on the image. In this way, a bigger digit should be obtained, with filled holes. Consequently, another way to differentiate between the digits with holes and the digits without holes may be to compute the difference between the area of the dilated digit and the area with the original digit. In this way, the difference would be smaller for the digits without holes and bigger for the digit with holes. But, before computing the difference, I normalised the ares in the same way I did with the perimeter.

Lastly but not least, I have computed the number of holes as the number of objects from the connected components in the complemented image - 1.

I also normalised this number by dividing it with 2 (as the digits can have maximum 2 holes inside).

4.3 Classifier

As far as the classifier is concerned, I have implemented a KNN with only one nearest neighbour, as we only need one output value.

The classification data is just a matrix constructed by concatenating the X features and the Y outputs. The concatenation was done in such a manner that the matrix has the features on the first rows and the class (-1 or 1) on the last row. Therefore, the training function has the following code.

```
function classification_data = class_train(X, Y)
    classification_data = [X; Y];
end
```

Listing 9: training function

For KNN, the basic idea is to compute the Euclidean distance for each x from the classification data set and each x from the training data set, respectively testing data set. Then the distances are summed. For every column in the classification data, we store the sum of distances. Afterwards, the array with the sum of distances is sorted, and the index of the smallest value is extracted. Lastly, the corresponding class of the index is stored in the output variable.

In the following pages, the code for the OCR system can be observed.

```
function S = im2segment(im)
    H = fspecial('gaussian'); % gaussian filter
    im = imfilter(im, H);

    level = 43;
    im = im > level; % threshold for binarization

    im= bwmorph(im,'fill'); %fill isolated interior pixels
    se =  strel('square', 1);
    im = imdilate(im, se); % dilation
    im = bwpropfilt(im,'perimeter',5); %keep only the 5
   objects with the largest perimeter
    im = imerode(im, se); % erosion
    im= bwmorph(im,'fill'); %fill isolated interior pixels
```

```
14
15      [L, num] = bwlabel(im);% extract the labels and the
        numbers of 8 connected components
16      noSegments = num;     % number of segments
17
18      m = size(im,1);
19      n = size(im,2);
20
21      S = cell(1,noSegments); %array cell with segments.
22      for i = 1:noSegments
23          segment = zeros(m,n);
24          segment(i == L) = 1;
25          S{i}= segment;
26      end
27  end
```

Listing 10: Segmentation

```
1   function features = segment2features(I)
2       %image having the same size as the bounding box of the
        region
3       digit_Img = regionprops(I,'all').Image;
4       % resize to standard size
5       digit_Img = imresize(digit_Img, [20, 20]);
6       %extract the size of the image, so they can be used for
        normalization
7       [m,n] = size(digit_Img);
8
9       %extract features
10      holes = bwconncomp(~I);
11      holes_no = holes.NumObjects - 1;
12
13      props = regionprops(digit_Img, 'all');
14
15      C = props.Centroid;
16      Cx = C(1);
17      Cy = C(2);
18      Circ = props.Circularity;
19      P = props.Perimeter;
20      EN = props.EulerNumber;
21      E = props.Eccentricity;
22      A = props.Area;
23      se = strel('sphere',5);
24      I_filled = imdilate(digit_Img, se);
25      [m2,n2] = size(I_filled);
26      props_filled = regionprops(I_filled, 'all');
```

```
    A_filled = props_filled.Area;
    diff_filled = A_filled/(m2*n2) - A/(m*n);
    O = props.Orientation;

    features = [O/180, diff_filled, E, P/(m*n), EN, Circ, Cx/n, Cy/m, holes_no/2];
end
```

Listing 11: Features

```
function y = classify(x, classification_data)
    % 1 KNN implementation
    [m,n] = size(classification_data);
    [mx,nx] = size(x);
    dist_sum = ones(1,n);
    %for each column in the classification data
    %and each column in x
    %compute the euclidian distance between the x values
    for i = 1:n
        sum = 0;
        for j = 1:mx
            %euclidian distance
            dist = sqrt((x(j,1)-classification_data(j,i))^2);
            %sum the distances
            sum = sum + dist;
        end
        %construct the array with the sum of distances
        %it will have the same number of columns as the classification data
        dist_sum(1,i) = sum;
    end
    %sort the array with the sum of distances
    [min_dist,index] = sort(dist_sum,'ascend');
    %use KNN wth only one neighbour
    k = 1;
    %extract the index of the mnmal value
    index = index(1:k);
    %select the corresponding result for the index
    y_coresp = classification_data(m,index);
    y= y_coresp;
end
```

Listing 12: Classifier

References

[1] M. Oskarsson, Image Analysis Course (2021), Lund University.
[2] Mathworks, https://www.mathworks.com/help/images/ref/regionprops.html