Scene Segmentation and Interpretation Coursework 1. *Image Segmentation*

Miroslav Radojević, Guillaume Lemaître
Universitat de Girona

***Abstract*—Segmentation is an algorithm that subdivides an image into its meaningful constituent regions or objects. Depending on the approach, segmenting algorithms use basic properties of image intensity values - discontinuity and similarity. In this work, region growing segmentation method is implemented and tested as a segmentation tool for grey and color images.**

I. Introduction and problem definition

Region growing method groups pixels or subregions into larger regions based on the predefined criteria for growth. The basic approach is to start with a set of ”seed” points and from these grow regions by appending to each seed those neighbouring pixels that have predefined properties similar to the seed. Intensity of grey level or color may be used to define similarity criteria [1]. Common approaches for color image segmentation are clustering algorithms such as k-means or Mixture of Principal Components, however, these algorithms do not take spatial information into account. Furthermore, clustering algorithms require prior information regarding number of clusters, which is a difficult or ambiguous task. An alternative set of algorithms exists which uses color similarity or intensity similarity and a region-growing approach to spatial information. Region growing is based on the following principles. The algorithm starts with a seed pixel, examines local pixels around it, determines the most similar one, which is then included in the region if it meets certain criteria. This process is followed until no more pixels can be added. The definition of similarity may be set in any number of different ways. Region growing algorithms have been used mostly in the analysis of grey-scale images, however, segmenting a color image using adequate properties can be accomplished successfully [2].

The task is to implement region growing on given set of grey-level and color pictures. Segmentation is done sequentially, each time examining neighbouring pixels of a region, that starts growing from ”seed” points. Those neighbours that are satisfying criteria of similarity are added to the region, leaving their own neighbours in the queue for further examination (Figure II). Neighbourhood is defined as 8-neighbourhood or 4-neighbourhood, depending on connection type. This way, region growing satisfies another principle of region-based segmentation - connectivity. In each case, the segmentation results should strongly be determined by a tuning parameter which defines aggregation criteria threshold.

II. Algorithm analysis

Region growing is designed so that it starts from specified seed points supplied as arguments. In case seed point are omitted at the input, default value for starting point is (0, 0). An algorithm that would successfully determine seed points

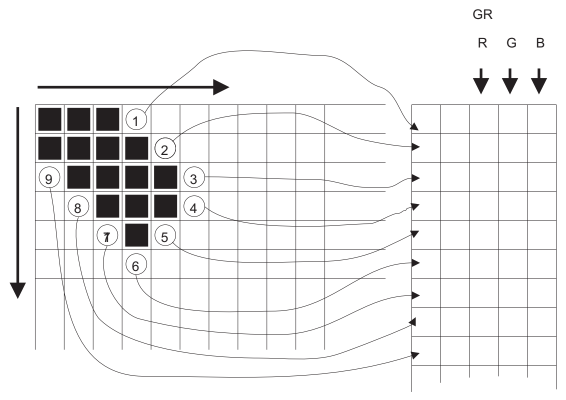


Fig. 1. Algorithm description scheme

would affect the outcome of the algorithm execution and quality of segmentation. Outcome depends on position and number of ”seeds”. Idea is that segmentation starts from points of interest and sequentially, pixel by pixel, expands the region by adding those pixels that satisfy the criteria of similarity (Figure II). Homogeneity criteria for colour image segmentation can be applied by using different colour spaces and different metrics. Criteria for similarity can be calculated using features, such as color intensity (red, green and blue components), hue, saturation, luminance or chrominance [2] and appropriate calculation, for instance, statistics, such as mean or standard deviation, or metrics distance. Calculated value is compared with threshold in order to obtain condition.

A. Computation of mean and standard deviation

In this section, we will present the calculation of mean and standard deviation. To compute these statistics, we used rapid calculation methods allowing to compute mean and standard deviation only with the last value known of this parameters.

1) Rapid computation of mean: We compute the mean value with a recursive method as follow:

$$\bar{m}_{i,c} = \bar{m}_{i-1,c} + \frac{(I_{(x,y,c)} - \bar{m}_{i-1,c})}{nb_p} \tag{1}$$

with initial condition

$$\bar{m}_{0,c} = 0 \tag{2}$$

where nb_p is the number of pixels, $I_{(x,y,c)}$ is the intensity of the pixel in coordinates x and y for the channel c, $\bar{m}_{i,c}$ is the mean value of the channel c at the instant i and $\bar{m}_{i-1,c}$ is the mean value of the channel c at the instant $i-1$.

2) Rapid computation of standard deviation: We compute the standard deviation value with a recursive method as follow:

$$std_{i,c} = \sqrt{\frac{varq_i}{nb_p}} \tag{3}$$

where:

$$\begin{aligned} varq_{i,c} &= varq_{i-1,c} \\ &+\frac{nb_p - 1}{nb_p} \times (I_{(x,y,c)} - \bar{m}_{i-1,c})^2 \end{aligned} \tag{4}$$

with initial condition

$$varq_{0,c} = 0 \tag{5}$$

where nb_p is the number of pixels, $I_{(x,y,c)}$ is the intensity of the pixel in coordinates x and y for the channel c and $std_{i,c}$ is the standard deviation at time i for the channel c.

B. Criteria

In this section, we will present different criteria that we used to decide if a pixel belongs or not to a region. First, we will present the criterion based on ”Euclidean distance”. Then, we will introduce a method using the mean value and a threshold. We will conclude with a criterion based on the standard deviation value of the region and the mean value of the region.

1) Euclidean distance criterion: We compute the ”Euclidean distance” between the value of the pixel and the mean value of the region to know if this pixel belongs to the region. We can formulate this criterion as follow:

$$\begin{aligned} D^2_{p,\bar{m}} &= (I_{(x,y,R)} - \bar{m}_{r,R})^2 \\ &+(I_{(x,y,G)} - \bar{m}_{r,G})^2 \\ &+(I_{(x,y,B)} - \bar{m}_{r,B})^2 \end{aligned} \tag{6}$$

where $I_{(x,y,c)}$ is the intensity of the pixel at the coordinates x and y for the channel c and $\bar{m}_{r,c}$ is the mean value of the region r of the channel c. The following equation defines the belonging criterion of a pixel to a region:

$$\begin{aligned} D_{p,\bar{m}} &\leq threshold \Rightarrow \text{pixel belongs to R} \\ D_{p,\bar{m}} &\geq threshold \Rightarrow \text{pixel does not belong to R} \end{aligned} \tag{7}$$

2) Mean value criterion: We call mean value criterion, a criterion based on the computation of the mean value of the region. In fact, we assume that a pixel belongs to the region if the value of the pixel is inside an interval defined by the mean value and a coefficient chosen by the user. The figure 2 present this criterion for one dimension. We work in a three dimensions space with red, green and blue. Hence, we can formulate the criterion as follow:

$$\begin{aligned} (I_{(x,y,1)} &\leq \bar{m}_{r,1} + k)\&\&(I_{(x,y,1)} &\geq \bar{m}_{r,1} - k) \\ (I_{(x,y,2)} &\leq \bar{m}_{r,2} + k)\&\&(I_{(x,y,2)} &\geq \bar{m}_{r,2} - k) \\ (I_{(x,y,3)} &\leq \bar{m}_{r,3} + k)\&\&(I_{(x,y,3)} &\geq \bar{m}_{r,3} - k) \end{aligned} \tag{8}$$

where $I_{(x,y,c)}$ is the intensity of the pixel at the coordinates x and y for the channel c and $\bar{m}_{r,c}$ is the mean value of the region r of the channel c. k is a threshold value defined by the user. To compute the mean, we use the rapid computation method shown in the section II-A.

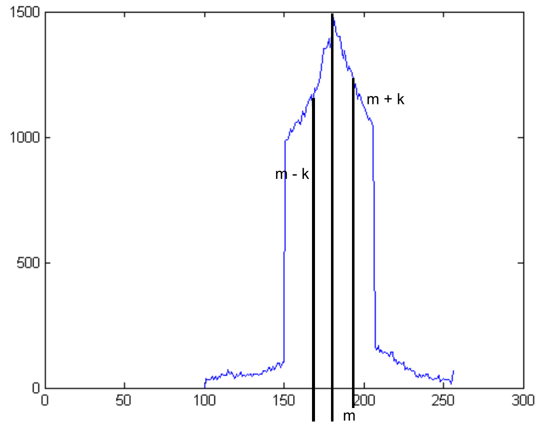


Fig. 2. Example of the mean value criterion for one dimension

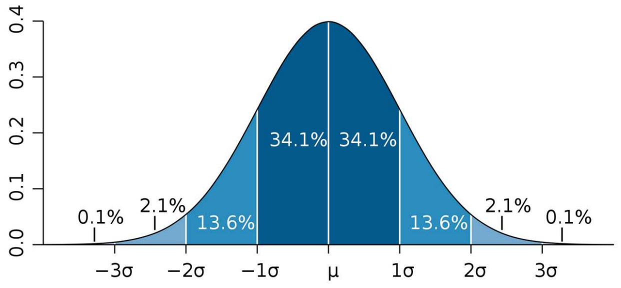


Fig. 3. Exemple of the standard deviation criterion for one dimension

3) Standard deviation criterion: We call standard deviation criterion, a criterion based on the computation of the mean value and standard deviation of the region. In fact, we assume that a pixel belongs to the region if the value is inside an interval defined by the mean value and k times the standard deviation of the region where k is defined by the user. The figure 3 presents this criterion for one dimension. We work in a three dimensions space with red, green and blue. Hence, we can formulate the criterion as follow:

$$\begin{aligned} (I_{(x,y,1)} &\leq \bar{m}_{r,1} + k \times std_{r,1}) \\ \&\&(I_{(x,y,1)} &\geq \bar{m}_{r,1} - k \times std_{r,1}) \\ (I_{(x,y,2)} &\leq \bar{m}_{r,2} + k \times std_{r,2}) \\ \&\&(I_{(x,y,2)} &\geq \bar{m}_{r,2} - k \times std_{r,2}) \\ (I_{(x,y,3)} &\leq \bar{m}_{r,3} + k \times std_{r,3}) \\ \&\&(I_{(x,y,3)} &\geq \bar{m}_{r,3} - k \times std_{r,3}) \end{aligned} \tag{9}$$

where $I_{(x,y,c)}$ is the intensity of the pixel at the coordinates x and y for the channel c and $\bar{m}_{r,c}$ is the mean value of the region r of the channel c. k is a threshold value defined by the user and $std_{r,c}$ is the standard deviation of the region r of the channel c To compute the mean and the standard deviation, we use the rapid computation method shown in the section II-A.

III. Design and implementation of the solution

Matlab was used to implement all algorithm of this assignment. The code of these implementations is available inside the Appendix A. We implemented four main functions allowing to perform the segmentation. These four functions are:

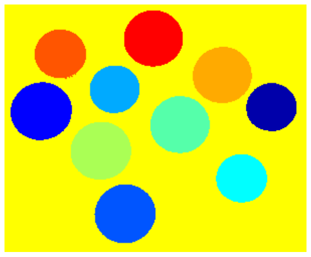
- Region growing: this function allows to segment the image using region based method with a 4 neighbourhood connectivity. The output of this function is an image segmented.
- Median filter function with window 3 by 3: as we will see in the discussion section, the image segmented return by the region growing algorithm is not perfect and present spots which can be consider like salt and peppers noise. To remove this noise, median filter is the best filter. We implemented two versions of this filter. One will be with a window size 3 by 3 and another with a window size 9 by 9.
- Creation of histograms. To evaluate the accuracy of the algorithm, we used a function to create histogram of the main regions of the segmented image. These histograms give a lot of information regarding the regions and if the algorithm performs correctly.

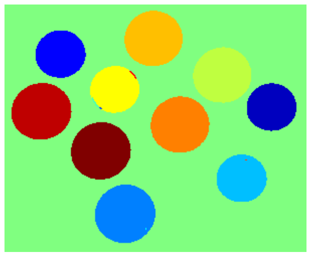
We will describe the design of the region growing function which is the main function of the algorithm. First, we start in a seed. We find the neighbours of the seed and put it in a queue. We will check the first neighbour in the queue and compute the criteria wanted (choice between several criteria presented in the part of the algorithm analysis). If the neighbour respects the criteria, it will belong to the queue and we will update the statistical parameters. If not, we will ignore this neighbour and marked like view. We will check the queue until this one will be empty. When the queue is empty, we search a new seed, which is the first pixel on the image that it is not labelled and start a new region.

IV. Results analysis

A. Segmentation results

(a) Original grey-level image *coins.jpg*


(b) After segmentation using m and σ, $Coeff = 6.1$, $t_{exec} = 1.6s$, $11 segments$


(c) After segmentation using m and threshold, $Coeff = 75$, $t_{exec} = 1.7s$, $16 segments$

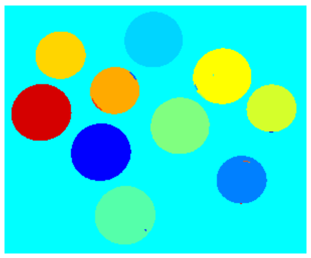
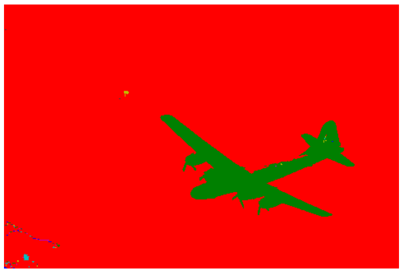
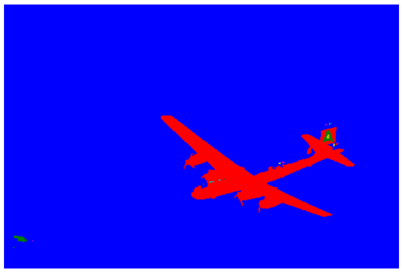

(d) After segmentation using euclidian distance, $Coeff = 70$, $t_{exec} = 1.8s$, $21 segments$

Fig. 4. Results of region growing segmentation for *coins.jpg* image using [1, 1] as starting point

(a) Original color image *airplane.jpg*


(b) After segmentation using m and σ, $Coeff = 4.2$, $t_{exec} = 5.06s$, $76 segments$


(c) After segmentation using m and threshold, $Coeff = 70$, $t_{exec} = 4.03s$, $20 segments$

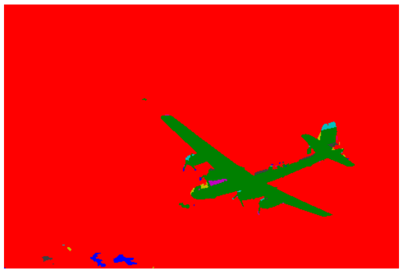
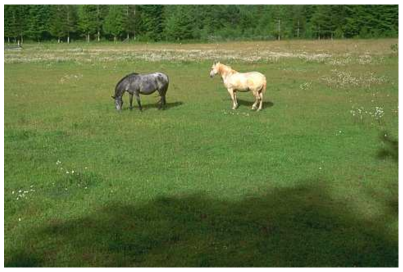
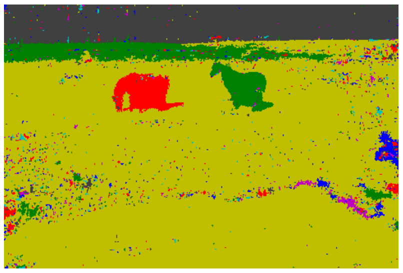
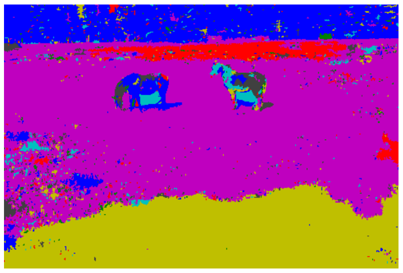

(d) After segmentation using euclidean distance, $Coeff = 75$, $t_{exec} = 5.8s$, $71 segments$

Fig. 5. Results of region growing segmentation for *airplane.jpg* image using [1, 1] as starting point


(a) Original color image *horses1.jpg*


(b) After segmentation using m and σ, $Coeff = 2.5$, $t_{exec} = 48.86s$, $2050 segments$


(c) After segmentation using m and threshold, $Coeff = 37$, $t_{exec} = 40.36s$, $1718 segments$

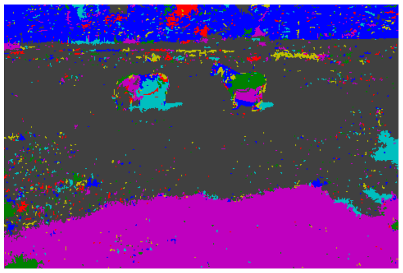
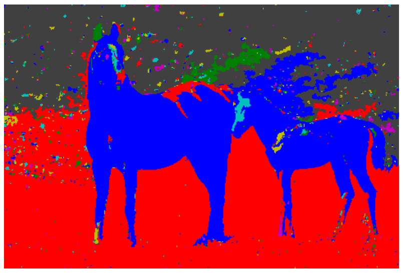
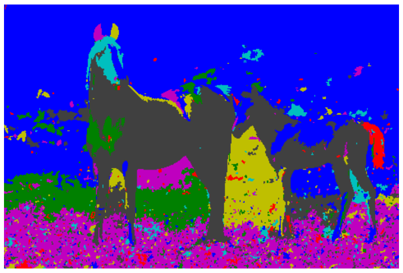

(d) After segmentation using euclidean distance, $Coeff = 47$, $t_{exec} = 52.55s$, $2282 segments$

Fig. 6. Results of region growing segmentation for *horses1.jpg* image using [1, 1] as starting point

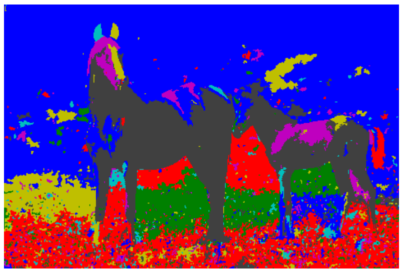
(a) Original color image *horses.jpg*



(b) After segmentation using m and σ, $Coeff = 2.5$, $t_{exec} = 32.67s$, $1724 segments$

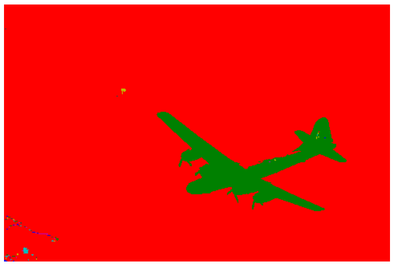


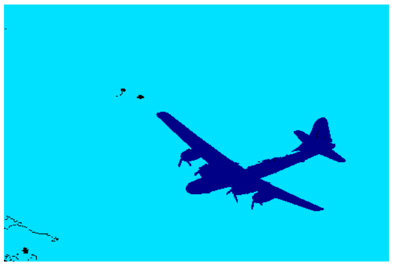
(c) After segmentation using m and threshold, $Coeff = 55$, $t_{exec} = 34.09s$, $1887 segments$

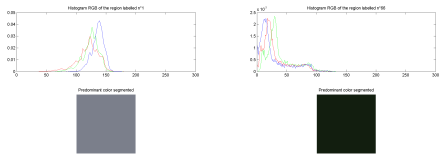


(d) After segmentation using euclidean distance, $Coeff = 70$, $t_{exec} = 35.69s$, $2046 segments$

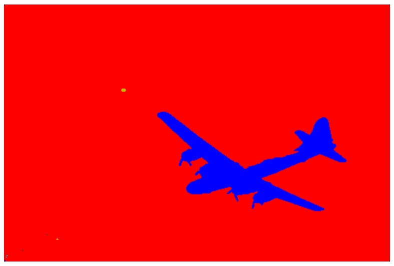
Fig. 7. Results of region growing segmentation for *horses.jpg* image using [1, 1] as starting point

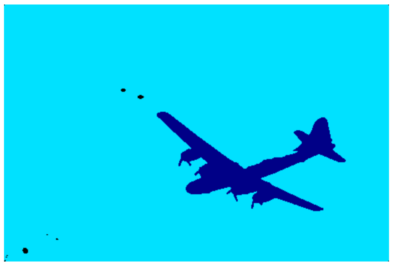

(a) Image *plane.jpg* after segmentation using region-growing

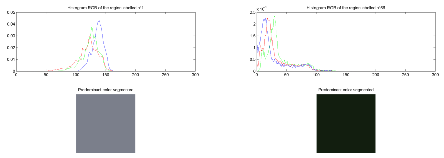

(b) Image after segmentation with segments containing more than 5% of total pixels extracted



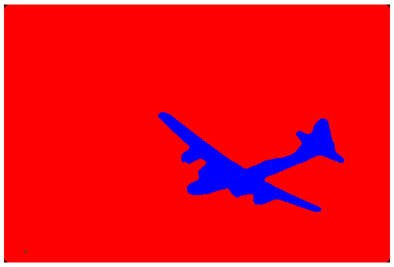
(c) Histograms of red, green and blue for each of the two main segments of the segmented image

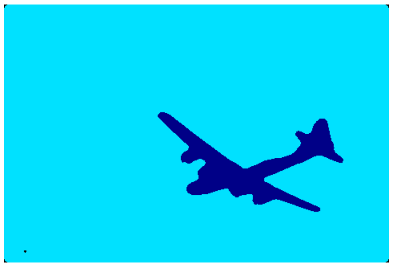

(d) Image *plane.jpg* after segmentation, filtered with $median3 \times 3$ filter

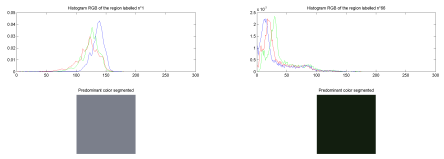

(e) $Median3 \times 3$ filtered image, with segments containing more than 5% of total pixels extracted



(f) Histograms of red, green and blue for each of the two main segments of the filtered image

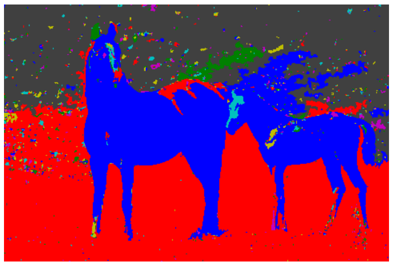

(g) Image *plane.jpg* after segmentation, filtered with $median9 \times 9$

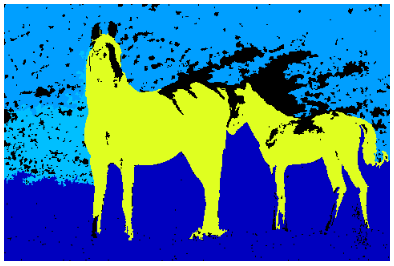

(h) $Median9 \times 9$ filtered image, with segments containing more than 5% of total pixels extracted

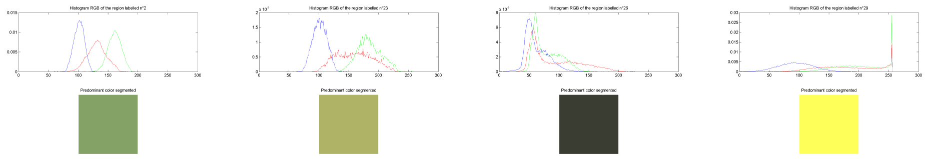


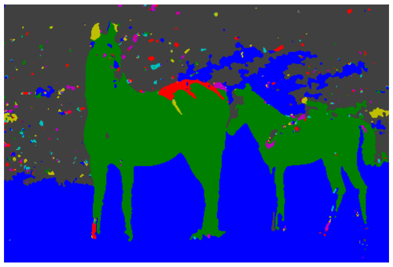
(i) Histograms of red, green and blue for each of the two main segments of the filtered image

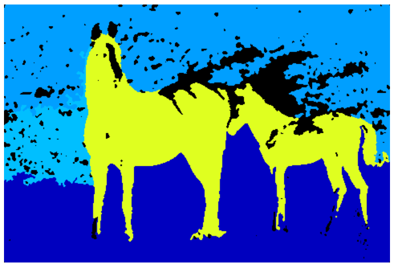
Fig. 8. Image *plane.jpg* segmentation results.

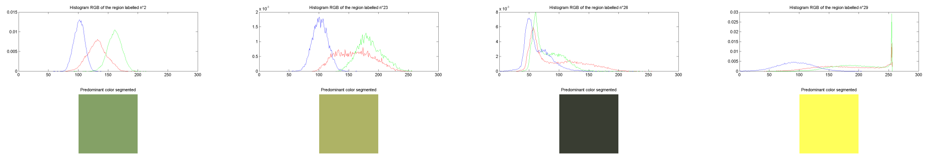

(a) Image *horses.jpg* after segmentation using region-growing

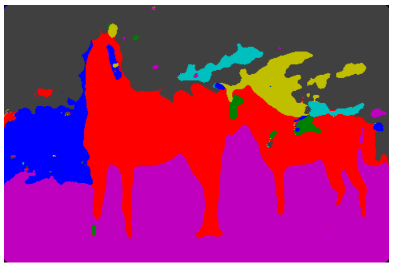

(b) Image after segmentation with segments containing more than 5% of total pixels extracted

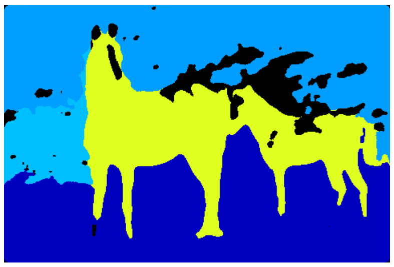

(c) Histograms of red, green and blue for each of the four main segments of the segmented image


(d) Image *horses.jpg* after segmentation, filtered with $median3 \times 3$ filter


(e) $Median3 \times 3$ filtered image, with segments containing more than 5% of total pixels extracted


(f) Histograms of red, green and blue for each of the four main segments of the filtered image


(g) Image *horses.jpg* after segmentation, filtered with $median9 \times 9$


(h) $Median9 \times 9$ filtered image, with segments containing more than 5% of total pixels extracted

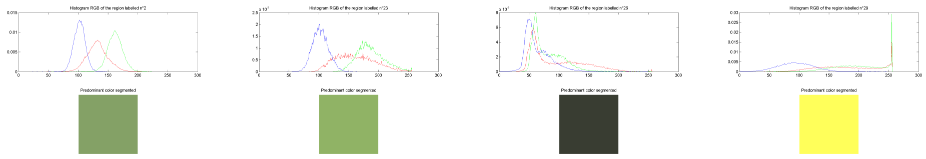

(i) Histograms of red, green and blue for each of the four main segments of the filtered image

Fig. 9. Image *horses.jpg* segmentation results.

B. Confusion matrices

GT Algo	Plane	No Plane
Plane	8622	88
No Plane	676	145015

TABLE I
CONFUSION MATRIX FOR SEGMENTED IMAGE *plane.jpg*, ACCURACY: 0.9951, SENSITIVITY: 0.9273, SPECIFICITY: 0.9994

GT Algo	Plane	No Plane
Plane	8642	94
No Plane	656	145009

TABLE II
CONFUSION MATRIX FOR $median3 \times 3$ FILTERED SEGMENTATION IMAGE OF *plane.jpg*, ACCURACY: 0.9951, SENSITIVITY: 0.9294, SPECIFICITY: 0.9994

GT Algo	Plane	No Plane
Plane	8507	169
No Plane	791	144934

TABLE III
CONFUSION MATRIX FOR $median9 \times 9$ FILTERED SEGMENTATION IMAGE OF *plane.jpg*, ACCURACY: 0.9938, SENSITIVITY: 0.9149, SPECIFICITY: 0.9988

GT Algo	Horses	No horses
Horses	32662	1179
No Horses	7351	113209

TABLE IV
CONFUSION MATRIX FOR SEGMENTED IMAGE *horses.jpg*, ACCURACY: 0.9448, SENSITIVITY: 0.8163, SPECIFICITY: 0.9897

GT Algo	Horses	No Horses
Horses	32904	1189
No Horses	7109	113199

TABLE V
CONFUSION MATRIX FOR $median3 \times 3$ FILTERED SEGMENTATION IMAGE OF *horses.jpg*, ACCURACY: 0.9463, SENSITIVITY: 0.8223, SPECIFICITY: 0.9896

GT Algo	Horses	No Horses
Horses	33432	1301
No Horses	6581	113087

TABLE VI
CONFUSION MATRIX FOR $median9 \times 9$ FILTERED SEGMENTATION IMAGE OF *plane.jpg*, ACCURACY: 0.9490, SENSITIVITY: 0.8355, SPECIFICITY: 0.9886

V. DISCUSSION

Segmentation results have shown that grey-level images segmentation does perform well and precise, although *coins.jpg* image was an easy task for all three versions of aggregation criteria (Figure 4), which is presented in Figures 4(b), 4(c), and 4(d). In case of a color image, segmenting becomes more complex and demanding. Segmentation using RGB values is at its best performance when using coefficient-multiplied standard deviation distance from the region's mean as a measure of similarity criteria (Figure 5). In this case, borders look smoother and authentic, which is due to taking into account statistics of the area and considering that diversity of the region when deciding the border, but it is computationally more expensive. Comparing mean value of the region with threshold parameter has slightly worse performance, since the information about standard deviation does not exist in similarity condition any more. Result is visible in significant increase of number of segments. Finally, euclidean distance performs worse, makes plenty of isolated spots - tiny numerous segments, and borders seem to be sharper. In this case, calculation of the mean or standard deviation is not necessary. Reason for it's bad speed performance is usage of square root function that is time consuming. Image *plane.jpg* tends to be easiest for color segmentation and results are by far the best and in this case considering both accuracy and time cost. However, the problem of isolated segments containing only several pixels remains as a disadvantage for all methods. Possible solution for this is implementation of the median filter while pre-processing (suggested as possible improvement) or post-processing (done in this assignment).

One of the important factors for quality of segmentation is input coefficient. Too low value of this parameter leads to over-segmentation and too high value leads to under-segmentation.

Each of the meaningful extracted segments in *plane.jpg* and *horses.jpg* is presented with histogram of its R, G and B values (Figures 8(c), 9(f), 9(i), 9(c), **??**, **??**). Histograms present the characteristics of the biggest regions of the image.

VI. IMPROVEMENTS

Very often the region growing algorithm generates too many segments (over-segmentation). There is possible to limit a number of segments by different additional pre-processing and post-processing procedures. Good example for pre-processing is the median filtering before image segmentation. This filtering procedure significantly decreases the number of regions in segmented image. Also, the region merging procedure as post-processing procedure can be used to avoid over-segmentation or remove small highlights from objects. This procedure locates all regions smaller than a given area, analyses their 4-connected neighbourhood and merges each region with most similar region from its neighbourhood.

VII. CONCLUSIONS

REFERENCES

[1] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[2] Henryk Palus and Damian Bereska. Region-based colour image segmentation, 1999.

Appendix A
Code

A. Main function

```
%Open the image
im = imread('28075.jpg');
%Initialisation of the seed
x_seedpoint = 1; y_seedpoint = 1;
%Show the input
figure, imshow(im); title('Input image');
%Fix the coefficient to perform the segmentation
Coeff = 2.5;
[segmented_image,statsvec] = Region_Growing4color2(im, x_seedpoint, y_seedpoint, Coeff, 's'); % using std
im_labelled = label2rgb(segmented_image, 'lines', 'k', 'shuffle');
figure, imshow(im_labelled); title('Segmented image');
imwrite(im_labelled,'Seg_Im_Reg_Grow.png','png');
sizevac = size(statsvec);

%%% Apply filtering
%%% Median 3x3
[medim3x3,nbregions3x3] = Median_Filter3(segmented_image);
figure, imshow(label2rgb(medim3x3, 'lines', 'k', 'shuffle'));
 title(['Segmented image filtered 3x3, number of regions: ', num2str(nbregions3x3)]);
imwrite(label2rgb(medim3x3, 'lines', 'k', 'shuffle'),'Seg_Med3x3.png','png');
%%% Median 9x9
[medim9x9,nbregions9x9] = Median_Filter9(segmented_image);
figure, imshow(label2rgb(medim9x9, 'lines', 'k', 'shuffle'));
 title(['Segmented image filtered 9x9, number of regions: ', num2str(nbregions9x9)]);
imwrite(label2rgb(medim9x9, 'lines', 'k', 'shuffle'),'Seg_Med9x9.png','png');
%%% Compute histogram of image
percthres = 0.005;
%%% Median image 9x9
[histR9x9,histg9x9,histB9x9,bigregim9x9,nbbigreg9x9] = Create_Histogram(im,medim9x9,nbregions9x9,percthres);
figure, imshow(label2rgb(bigregim9x9, 'jet', 'k', 'shuffle'));
 title(['Number of region: ', num2str(nbbigreg9x9), ' superior at ', num2str(percthres)]);
imwrite(label2rgb(bigregim9x9, 'jet', 'k', 'shuffle'),'Seg_Big_Reg_Med9x9.png','png');

%%% Median image 3x3
[histR3x3,histg3x3,histB3x3,bigregim3x3,nbbigreg3x3] = Create_Histogram(im,medim3x3,nbregions3x3,percthres);
figure, imshow(label2rgb(bigregim3x3, 'jet', 'k', 'shuffle'));
 title(['Number of region: ', num2str(nbbigreg3x3), ' superior at ', num2str(percthres)]);
imwrite(label2rgb(bigregim3x3, 'jet', 'k', 'shuffle'),'Seg_Big_Reg_Med3x3.png','png');

%%% Segmented Image
[histR,histG,histB,bigregim,nbbigreg] = Create_Histogram(im,segmented_image,sizevac(1)-1,percthres);
figure, imshow(label2rgb(bigregim, 'jet', 'k', 'shuffle'));
 title(['Number of region: ', num2str(nbbigreg), ' superior at ', num2str(percthres)]);
imwrite(label2rgb(bigregim, 'jet', 'k', 'shuffle'),'Seg_Big_Reg_Seg.png','png');
```

B. Algorithm of region growing

```
function [im_out,statsvec] = Region_Growing4color2(im_in, x, y, Coeff, type)
disp('processing color segmentation... 4-neighbourhood');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% this function performs "region growing" in an image im_in from a specified
% seedpoint (x,y)
% im_out = Region_Growing(im_in, x, y, Coeff, type)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% im_in : input image, with integer values of pixel color
% im_out : output image with segmented regions
% x, y : coordinates of the seedpoint position (if not given uses beginning point (1,1))
% Coeff : coefficient used for defining homogenity criteria of adding pixel to a region(defaults to 2)
% type : defines aggregation criteria - 's' using standard deviation
%                                       't' using threshold
%                                       'e' using euclidian distance
% statsvec : statistic vector regarding all regions
%            statsvec[index_region, meanR, meanG, meanB, stdR, stdG, stdB]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% the region is iteratively grown by comparing all unallocated neighbouring
% pixel values with the region values. Coeff is used as a parameter when defining measure of
% similarity, aggregation criteria.
% 't' mode - the difference between a pixel's intensity value and the region's
% mean, thresholded with Coeff
% 's' mode - check if the pixel's intensity value fits in the region's interval mean +/- Coeff * standard dev
% 'e' mode - measures if euclidian distance becomes more that threshold defined by Coeff
% if aggregation criteria measured this way is true, pixel is allocated to the respective region.
% growing of the region stops when the aggregation criteria becomes false,
% then the new region is begun
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% default argument values
if(exist('Coeff','var')==0), Coeff = 3.0; end
if(exist('x','var')==0), x = 1; end
if(exist('y','var')==0), y = 1; end

t = cputime; % measuring starting moment of execution time for segmentation
im_out = zeros(size(im_in(:, :, 1))); % output initialization
% initializing segmentation
[im_height im_width] = size(im_in(:, :, 1)); pixels_segmented = 0; label = 1;
% initialisation of the statistical vector
statsvec = []; anclabel = 0;
% loop that manages segmentation of the image, consisits of sequential
% segmentation of individual regions, works until all pixels are segmented
while (pixels_segmented<(im_height*im_width)),
    J = zeros(size(im_in(:, :, 1))); % matrix for avoiding creating double instances in queue
                                     % it resets each time a new region starts
    region_size = 0;
    % initialization of statistical parameters for red, green and blue
    region_mean =   0.0; region_std_q =   0.0; region_std =   0.0;
    region_mean_2 = 0.0; region_std_q_2 = 0.0; region_std_2 = 0.0;
    region_mean_3 = 0.0; region_std_q_3 = 0.0; region_std_3 = 0.0;
    % initialisation of statistical vector
    if(anclabel ≠ label)
        statsvec = [statsvec ;
        [label region_mean region_mean_2 region_mean_3 region_std region_std_2 region_std_3]];
    end
    anclabel = label;
    neighbour = [-1 0; 0 1; 1 0; 0 -1]; % 4 neighbourhood, implemented clockwise
    %%%%%%%%%%%%%%%%%%%%%%%%%%
    % Queue initialization
    %%%%%%%%%%%%%%%%%%%%%%%%%%
    Queue_size = im_height*im_width; Queue_end = 0; Queue_start = 0; Queue = zeros(Queue_size, 5);
    %%%%%%%%%%%%%%%%%%%%%%%%%%
    if isempty(x), [x, y] = find(im_out == 0, 1); end % if next region coordinates cannot be found,
                                                      % algorithm searches for the first available
                                                      % that is not segmented
    count = 0; % counter giving 2 iterations for standard deviation calculation to establish
    % loop that does segmentation of one individual region
    while (1)
        if ((count≤1) || homogenity_condition),
            count = count + 1; if count >1, count = 2; end % count enables execution two times regardless of
                                                           % homogenity condition so that standard deviation can
                                                           % start to converge without being interrupted by
                                                           % condition
            im_out(x, y) = label; % labelling the pixel: labels start from 1 and increment: 1, 2, 3...
            region_size = region_size + 1; % region size increases after labelling
            % keeping track of region size
            % iterative, recursive calculation of region mean and standard deviation
            % (better speed performance)
            % using values from previous iteration

            % first dimension - red for color images - mean and std recursion
            region_std_q  = region_std_q +
            (double(region_size - 1)/double(region_size)) * (double(im_in(x, y, 1)) - region_mean)^2;
            region_mean = region_mean + (double(im_in(x, y, 1)) - region_mean) / double(region_size);
            region_std = sqrt(double(region_std_q / (region_size)));
            statsvec(label,2) = region_mean;
            statsvec(label,5) = region_std;

            % second dimension - green for color images - mean and std recursion
            region_std_q_2  = region_std_q_2 +
            (double(region_size - 1)/double(region_size)) * (double(im_in(x, y, 2)) - region_mean_2)^2;
            region_mean_2 = region_mean_2 + (double(im_in(x, y, 2)) - region_mean_2) / double(region_size);
            region_std_2 = sqrt(double(region_std_q_2 / (region_size)));
            statsvec(label,3) = region_mean_2;
            statsvec(label,6) = region_std_2;

            % third dimension - blue for color images - mean and std recursion
            region_std_q_3  = region_std_q_3 +
            (double(region_size - 1)/double(region_size)) * (double(im_in(x, y, 3)) - region_mean_3)^2;
            region_mean_3 = region_mean_3 + (double(im_in(x, y, 3)) - region_mean_3) / double(region_size);
            region_std_3 = sqrt(double(region_std_q_3 / (region_size)));
            statsvec(label,4) = region_mean_3;
            statsvec(label,7) = region_std_3;

            surrounding_regions = [];
            for j = 1 : 4,
                xn = x + neighbour(j,1); yn = y + neighbour(j,2);
                inside = (xn≥1)&&(yn≥1)&&(xn≤im_height)&&(yn≤im_width);
                if(inside&&(im_out(xn,yn)==0)&&(J(xn, yn)==0))

                    % Queue entry - coordinates and R, G, B
                    Queue_end = Queue_end + 1;
                    Queue(Queue_end, 1) = xn; Queue(Queue_end, 2) = yn;
                    Queue(Queue_end, 3) = im_in(xn, yn, 1);
                    Queue(Queue_end, 4) = im_in(xn, yn, 2);
                    Queue(Queue_end, 5) = im_in(xn, yn, 3);

                    J(xn, yn) = 1; % keeping track of values that are entered in queue
                                   % so that they are not entered in queue again
                elseif (inside && (im_out(xn, yn) ≠ 0)),
                    %memorizing neighbouring pixels that were segmented
                    surrounding_regions = [surrounding_regions im_out(xn, yn)];
                end
            end
            pixels_segmented = pixels_segmented + 1;
        end
        Queue_start = Queue_start + 1;
        if Queue_end == 0, break; end % exits the loop if none of the first neighbours was entered to queue
                                      % queue was
%         if (Queue_start>Queue_end) && (im_out(x, y)==0),
%             x = double(Queue(Queue_end, 1)); y = double(Queue(Queue_end, 2));
%             break;
%         elseif (Queue_start>Queue_end)&& (im_out(x, y)≠0)
%             [x, y] = find(im_out == 0, 1);
%             break;
%         end
        if (Queue_start > Queue_end)
            if (im_out(x, y) == 0),
                x = double(Queue(Queue_end, 1)); y = double(Queue(Queue_end, 2));
            else
                [x, y] = find(im_out == 0, 1);
            end
            break;
        end
            pixel_value   = Queue(Queue_start, 3);
            pixel_value_2 = Queue(Queue_start, 4);
            pixel_value_3 = Queue(Queue_start, 5);
            x = double(Queue(Queue_start, 1)); y = double(Queue(Queue_start, 2));
            % aggregation criteria type
            if type == 't',
                % use Coeff as threshold
                homogenity_condition = ((pixel_value  ≤ region_mean  + Coeff  )
                &&(pixel_value  ≥ region_mean  - Coeff  )) && ...
                                       ((pixel_value_2≤ region_mean_2+ Coeff  )
                                       &&(pixel_value_2≥ region_mean_2- Coeff  )) && ...
                                       ((pixel_value_3≤ region_mean_3+ Coeff  )
                                       &&(pixel_value_3≥ region_mean_3- Coeff  ));
            else if type == 's',
                    % use Coeff as std multiplier
                    homogenity_condition = ((pixel_value  ≤ region_mean  + Coeff*region_std  )
                    &&(pixel_value  ≥ region_mean  - Coeff*region_std  )) && ...
                                           ((pixel_value_2≤ region_mean_2+ Coeff*region_std_2)
                                           &&(pixel_value_2≥ region_mean_2- Coeff*region_std_2)) && ...
                                           ((pixel_value_3≤ region_mean_3+ Coeff*region_std_3)
                                           &&(pixel_value_3≥ region_mean_3- Coeff*region_std_3));
                else
                    if type == 'e',
                        % use Coeff as threshold for euclidian distance
                        homogenity_condition = sqrt((pixel_value - region_mean)^2 +
                         (pixel_value_2 - region_mean_2)^2 + (pixel_value_3 - region_mean_3)^2) ≤ Coeff;
                    end
                end
            end

    end % while (1)
    if Queue_end == 0,
        im_out(x, y) = mode(surrounding_regions);
        label = label - 1;
        x = []; y = [];
    end
    label = label + 1;
end % while (pixels_segmented<(im_height*im_width))
e = cputime - t;
disp('finished');
fprintf(1, '\nElapsed time: %3.2f seconds', e);
fprintf(1, '\nNumber of segments: %i', label-1);
fprintf(1, '\nPixels segmented: %i (out of %i)\n\n', pixels_segmented, im_height*im_width);
```

C. Creation of histogram of the main region

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function to construct histogram of each region with a size suffisant
%%% which are a percentage of the size of the image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Input:
%%%     im: Original Image
%%%     segmented_image: Segmented image
%%%     nbregion_im_seg: number of region of the segmented image
%%%     percthres: Threshold to take only region superior to this
%%%     percentage (from 0 to 1)
%%% Out:
%%%     histR: Histogram Red of the image
%%%     histG: Histogram Green of the image
%%%     histB: Histogram Blue of the image
%%%     bigregionim: image contening only nbreg (biggest region)
%%%     nbreg: number of biggest region found
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [histR, histG, histB, bigregionim, nbreg] =
Create_Histogram(im,segmented_image,nbregion_im_seg,percthres)

%%% Compute histogram of each region
[heightim,widthim] = size(segmented_image);
%%% Allow to know the number and which are the different labels in the
%%% segmented image
vecreg = unique(segmented_image);
%%% Allocate histograms with a maximum number of nbregions
histR = zeros(256,nbregion_im_seg);
histG = zeros(256,nbregion_im_seg);
histB = zeros(256,nbregion_im_seg);
for i = 1:nbregion_im_seg
    for j = 1:heightim
        for k = 1:widthim
            if (vecreg(i) == segmented_image(j,k))
                histR(im(j,k,1)+1,i) = histR(im(j,k,1)+1,i) + 1;
                histG(im(j,k,2)+1,i) = histG(im(j,k,2)+1,i) + 1;
                histB(im(j,k,3)+1,i) = histB(im(j,k,3)+1,i) + 1;
            end
        end
    end
    histR(:,i) = histR(:,i) / (heightim*widthim);
    histG(:,i) = histG(:,i) / (heightim*widthim);
    histB(:,i) = histB(:,i) / (heightim*widthim);
end

bigregionim = zeros(size(segmented_image));
nbreg = 0;
for i = 1:nbregion_im_seg
    sumRegionR = sum(histR(:,i));
    sumRegionG = sum(histG(:,i));
    sumRegionB = sum(histB(:,i));
    if (sumRegionR > percthres)||(sumRegionG > percthres)||(sumRegionB > percthres)
        nbreg = nbreg + 1;
        figure;
        hold on;
        subplot(211)
        plot(histR(:,i),'-r');
        hold on;
        plot(histG(:,i),'-g');
        hold on;
        plot(histB(:,i),'-b');
        hold on;
        title(['Histogram RGB of the region labelled n ', num2str(vecreg(i))]);
        hold on;
        squareim = zeros(16,16,3);
        for j = 1:256
            if (histR(j,i) == max(histR(:,i)))
                val1 = j;
            end
            if (histG(j,i) == max(histG(:,i)))
                val2 = j;
            end
            if (histB(j,i) == max(histB(:,i)))
                val3 = j;
            end
        end
        for j = 1:16
            for k = 1:16
                squareim(j,k,1) = val1;
                squareim(j,k,2) = val2;
                squareim(j,k,3) = val3;
            end
        end
        hold on;
        subplot(212)
        imshow(uint8(squareim));title('Predominant color segmented');
        saveas(gcf,['Hist_Reg_',num2str(nbreg)],'png');
        for j = 1:heightim
            for k = 1:widthim
                if(vecreg(i) == segmented_image(j,k))
                    bigregionim(j,k) = segmented_image(j,k);
                end
            end
        end
    end
end
```

D. Median filter with window 3 by 3

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function to apply median filter with window 3x3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Input:
%%%     inputim: input image (1 channel only)
%%% Out:
%%%     fimage: image filtered (1 channel only)
%%%     nbregions: number of regions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [fimage, nbregions] = Median_Filter3(inputim)

%%% Apply filter with window 3 by 3
fimage = medfilt2(inputim);

%%% Count the number of regions
vecregions = unique(fimage);
[nbregions, depth] = size(vecregions);
```

E. Median filter with window 9 by 9

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function to apply median filter with window 9x9
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Input:
%%%     inputim: input image (1 channel only)
%%% Out:
%%%     fimage: image filtered (1 channel only)
%%%     nbregions: number of regions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [fimage, nbregions] = Median_Filter9(inputim)

%%% Apply filter with window 3 by 3
fimage = medfilt2(inputim, [9 9]);

%%% Count the number of regions
vecregions = unique(fimage);
[nbregions, depth] = size(vecregions);
```