Image Processing Assignment - I

Topic : Image Segmentation

CSE-A
Anup Das (106116011)
Neel Akash (106116059)

Problem Statement :

Perform **Image Segmentation** using 3 techniques from each criteria, over a simple image without using inbuilt functions.

Algorithm :

This codebase includes usage of

- Prewitt Method for **Edge detection**
- Global Method for **Thresholding**
- Region Growing Method for **Region Segmentation**

Done using MATLAB, used sample world.png as sample image input for all.
Github : github.com/theofficialneel/img-segmentation-examples

[A] Edge Detection by Prewitt Method :

Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. Edge detection is used for image segmentation and data extraction in areas such as image processing, computer vision, and machine vision.

- **Approach :**

For edge detection, we take the help of convolution: Convolution = I * m where **I** is the image, **m** is the mask and ***** is the convolutional operator. To perform convolution on an image following steps are required:-

- ➢ Flip the mask horizontally and then vertically. This will result in 180-degree rotation of an image.
- ➢ Slide the mask onto the image such that every pixel in the image coincides with the center of the mask at least once.
- ➢ Multiply the corresponding elements with the pixel values below it and then add them.
- ➢ Repeat this procedure until all pixel values of the image have been calculated for updation.

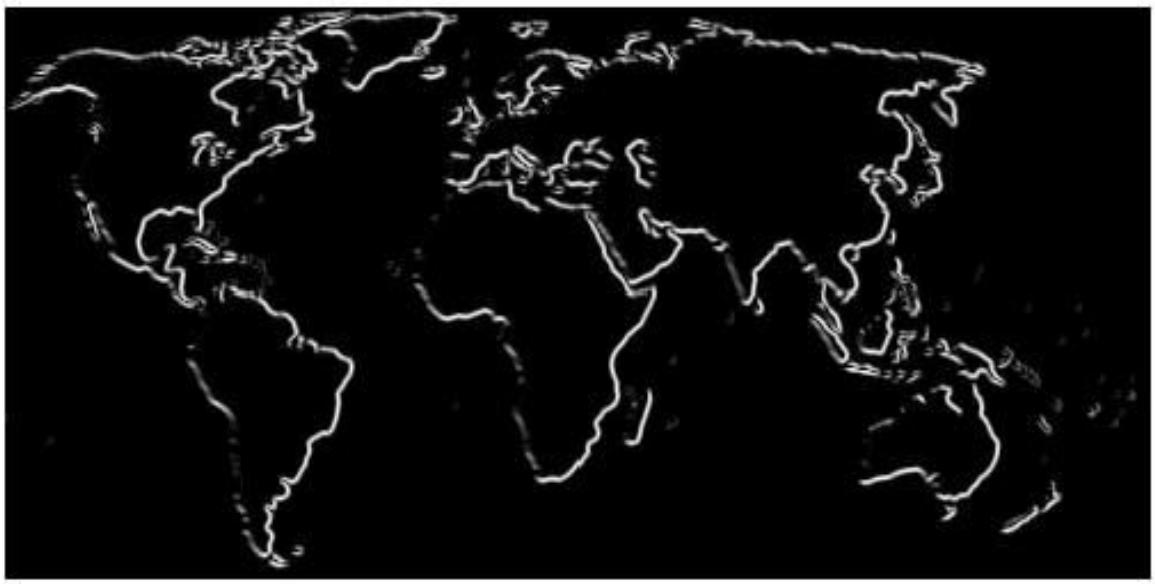
- **Code :**

```
function edgeDetect
	clear; clc;
	% Convert to grayscale and then scalar
	I=double(rgb2gray(imread('world.png')));
	In=I;
	% Create the Prewitt masks
	mask1=[1, 0, -1;1, 0, -1;1, 0, -1]; % Vertical
	mask2=[1, 1, 1;0, 0, 0;-1, -1, -1]; % Horizontal
	mask3=[0, -1, -1;1, 0, -1;1, 1, 0]; % -45 deg
	mask4=[1, 1, 0;1, 0, -1;0, -1, -1]; % +45 deg

	% Flip masks horizontally and vertically
	mask1=flipud(mask1);
	mask1=fliplr(mask1);
	mask2=flipud(mask2);
	mask2=fliplr(mask2);
	mask3=flipud(mask3);
	mask3=fliplr(mask3);
	mask4=flipud(mask4);
	mask4=fliplr(mask4);

	for i=2:size(I, 1)-1
		for j=2:size(I, 2)-1
		% mask 1
		neighbour_matrix1=mask1.*In(i-1:i+1, j-1:j+1);
		avg_value1=sum(neighbour_matrix1(:));

		% mask 2
		neighbour_matrix2=mask2.*In(i-1:i+1, j-1:j+1);
```

```
            avg_value2=sum(neighbour_matrix2(:));


            % mask 3
            neighbour_matrix3=mask3.*In(i-1:i+1, j-1:j+1);
            avg_value3=sum(neighbour_matrix3(:));


            % mask 4
            neighbour_matrix4=mask4.*In(i-1:i+1, j-1:j+1);
            avg_value4=sum(neighbour_matrix4(:));


            %using max function for detection of final edges
            I(i, j)=max([avg_value1, avg_value2, avg_value3,
            avg_value4]);
            end
      end
      figure, imshow(uint8(I));
end
```

- **Input Image :**

- **Output Image :**



[B] Thresholding by Global Method :

An image processing method that creates a bitonal (aka binary) image based on setting a threshold value on the pixel intensity of the original image. While most commonly applied to grayscale images, it can also be applied to color images.

- **Approach :**
  To perform the thresholding I followed these steps:
  - ➢ Reshape the 2 dimensional grayscale image to 1 dimensional.
  - ➢ Find the histogram of the image using the 'hist' function.
  - ➢ Initialize a matrix with values from 0 to 255

- ➢ Find the weight , mean and the variance for the foreground and background.
- ➢ Calculate weight of foreground* variance of foreground + weight of background* variance of background.

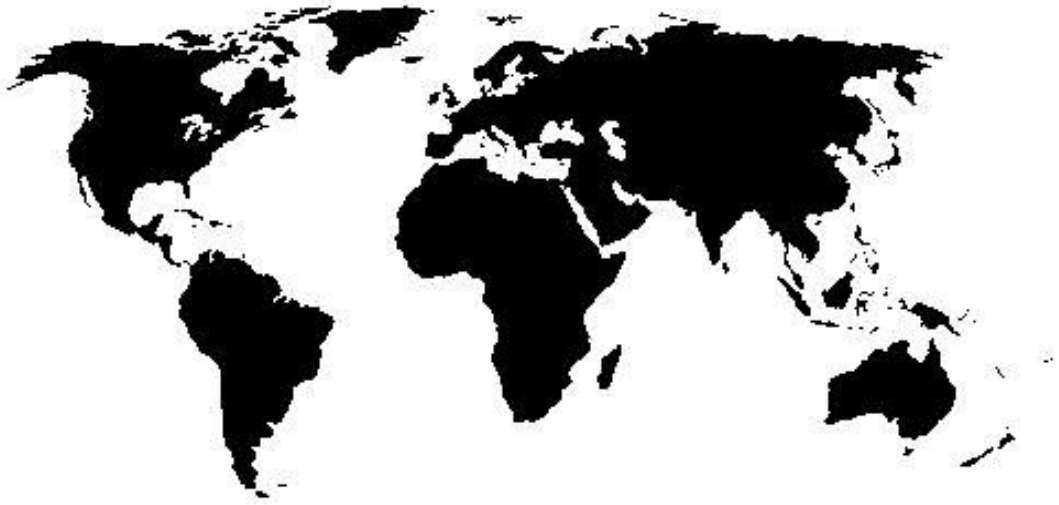
- **Code :**

```
function globalThreshold
      clear; clc;
      % Global Threshold Segmentation
      global H Index;
      B=imread('world.png');

      V=reshape(B,[],1);
      G=hist(V,0:255);
      H=reshape(G,[],1);

      Ind=0:255;
      Index=reshape(Ind,[],1);
      result=zeros(size([1 256]));

      for i=0:255
            % Custom calculate function defined below
            [wbk,varbk]=calculate(1,i);
            [wfg,varfg]=calculate(i+1,255);
            result(i+1)=(wbk*varbk)+(wfg*varfg);
      end

      [~,val]=min(result);
      tval=(val-1)/256;

      bin_im=im2bw(B,tval);
      figure, imshow(bin_im);

      function [weight,var]=calculate(m,n)
            %Weight Calculation
            weight=sum(H(m:n))/sum(H);

            %Mean Calculation
            value=H(m:n).*Index(m:n);
```

```
            total=sum(value);
            mean=total/sum(H(m:n));

            if(isnan(mean))
                  mean=0;
            end

            %Variance calculation.
            value2=(Index(m:n)-mean).^2;
            numer=sum(value2.*H(m:n));
            var=numer/sum(H(m:n));

            if(isnan(var))
                  var=0;
            end
      end
end
```

- **Input Image :**

- **Output Image :**



[C] Region-based segmentation (Region Growing Method) :

Region growing is a simple region-based image segmentation method. It is also classified as a pixel-based image segmentation method since it involves the selection of initial seed points.

- **Approach :**

This approach to segmentation examines neighboring pixels of initial seed points and determines whether the pixel neighbors should be added to the region. The process is iterated on, in the same manner as general data clustering algorithms. A general discussion of the region growing algorithm is described below. The basic formulation is

- ➢ that the segmentation must be complete; that is, every pixel must be in a region.
- ➢ requires that points in a region must be connected in some predefined sense.
- ➢ indicates that the regions must be disjoint.

- ➢ deals with the properties that must be satisfied by the pixels in a segmented region.
- ➢ indicates that region R i and R j are different in the sense of predicate P.

- **Code :**

Note : This code involves the user to click a point as seed, and then will take a couple of seconds to compile the region

```
function regionGrow
      clear; clc;
      path='world.png';
      I = im2double(imread(path));
      [y,x] = getpts(get(imshow(path),'Parent'));
      J = growFunction(I,x,y,0.2);
      figure, imshow(I+J);
end

function J = growFunction(I,x,y,reg_maxdist)
      if(exist('reg_maxdist','var')==0), reg_maxdist=0.2; end
      if(exist('y','var')==0), figure, imshow(I,[]); [y,x]=getpts;
y=round(y(1)); x=round(x(1)); end

      x=floor(x);y=floor(y); % Need only positive integers
      J = zeros(size(I)); % Output
      Isizes = size(I); % Dimensions of input image

      reg_mean = I(x,y); % The mean of the segmented region
      reg_size = 1; % Number of pixels in region

      % Free memory to store neighbours of the (segmented) region
      neg_free = 10000; neg_pos=0;
      neg_list = zeros(neg_free,3);

      pixdist=0; % Distance of the region newest pixel to the regio mean

      % Neighbor locations (footprint)
      neigb=[-1 0; 1 0; 0 -1;0 1];

      % Start regiogrowing until distance between regio and posible new
pixels become
      % higher than a certain treshold
      while(pixdist<reg_maxdist&&reg_size<numel(I))

      % Add new neighbors pixels
      for j=1:4
            % Calculate the neighbour coordinate
            xn = x +neigb(j,1); yn = y +neigb(j,2);

            % Check if neighbour is inside or outside the image
            ins=(xn>=1)&&(yn>=1)&&(xn<=Isizes(1))&&(yn<=Isizes(2));
```

```
            % Add neighbor if inside and not already part of the
segmented area
            if(ins&&(J(xn,yn)==0))
                  neg_pos = neg_pos+1;
                  neg_list(neg_pos,:) = [xn yn I(xn,yn)]; J(xn,yn)=1;
            end
      end

      % Add a new block of free memory
      if(neg_pos+10>neg_free), neg_free=neg_free+10000;
neg_list((neg_pos+1):neg_free,:)=0; end

      % Add pixel with intensity nearest to the mean of the region, to
the region
      dist = abs(neg_list(1:neg_pos,3)-reg_mean);
      [pixdist, index] = min(dist);
      J(x,y)=2; reg_size=reg_size+1;

      % Calculate the new mean of the region
      reg_mean= (reg_mean*reg_size + neg_list(index,3))/(reg_size+1);

      % Save the x and y coordinates of the pixel
      x = neg_list(index,1); y = neg_list(index,2);

      % Remove the pixel from the neighbour (check) list
      neg_list(index,:)=neg_list(neg_pos,:); neg_pos=neg_pos-1;
      end
end
```

- **Input Image :**

- **Seed Point Clicked :**

- **Final Output Image (Region Grow) :**

Result :

Hence shown the process of Image segmentation over the 3 categories on a simple image using MATLAB.