Narges Sadat Seyed Haeri – 810100165

Signals and systems

CA2 report

- ***Part 1:***

1. we use uigetfile function, It allows you to choose an image file with a specific set of file extensions, then we concatenate path and file to create the full path to the selected image and stores it in the variable s. imread is used to read the selected image file into the variable picture:

```
[file,path]=uigetfile({'*.jpg;*.bmp;*.png;*.tif'},'Choose an image');
s=[path,file];
picture=imread(s);
```

2. The imersize assigns the result of resizing the picture to have dimensions 300x500 to the variable picture:

```
picture=imresize(picture,[300 500]);
figure
subplot(1,2,2)
imshow(picture)
```

3. The mygray function checks if the input image picture has three color channels (indicating it's a color image) by using the size function. If the image has three channels (R, G, and B), it proceeds with the conversion.For color images, it calculates the grayscale value for each pixel using the formula that we were given.If the input image is already grayscale ,the function simply returns the input image without any further processing.Finally, the resulting grayscale image is cast to an unsigned 8-bit integer data type (uint8) and returned.

```
function grayimage = mygrayfunc(picture)
    if size(picture, 3) == 3
        grayimage = 0.299 * picture(:,:,1) + 0.587 * picture(:,:,2) + 0.114 * picture(:,:,3);
    else
        grayimage = picture;
    end
    grayimage = uint8(grayimage);
end
```

4. The mybinary function takes the input image, which is a grayscale image, and the threshold value as its arguments.It performs a simple binary conversion by comparing each pixel value in the input image to the threshold. If a pixel's value is less than the threshold, it is set to 1 (white in binary). If the pixel's value is greater than or equal to the threshold, it is set to 0 (black in binary).The result is a binary image with the same dimensions as the input image, where pixel values are either 0 or 1.

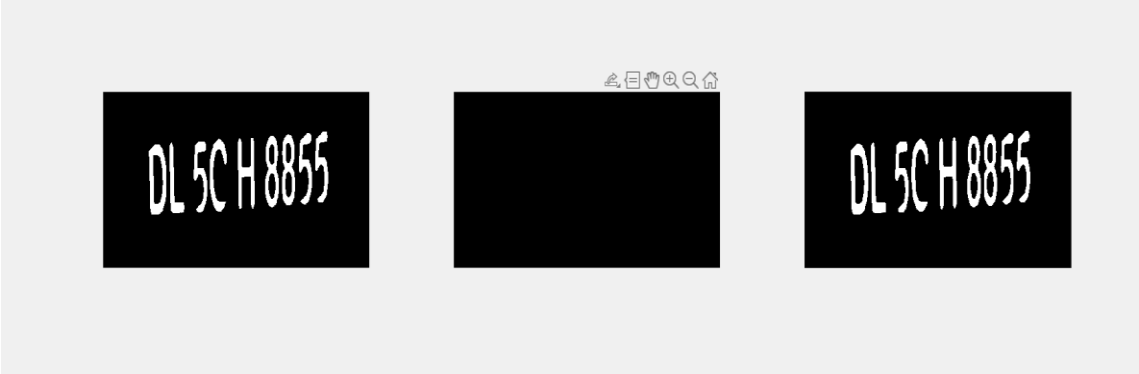
```
function binaryImage = mybinaryfunc(inputImage, threshold)
    binaryImage = inputImage < threshold;
end
```

Picture after changes made from part 1 to 4:

5. First we should implement mysegmentation function from part 6, Now this function myremovecom takes a binary image (BW) and a minimum area threshold (minArea) as input and removes connected components in the binary image that have an area smaller than the specified threshold. The function first calls mysegmentation function to label the connected components in the binary image BW. This function labels each connected component with a unique integer value, and the labeled image is stored in the variable **L** .An empty list keepLabels is initialized to store the labels of the connected components that should be kept in the cleaned binary image.The function then iterates through the labeled components, starting from label 1 up to the maximum label value found in the labeled image (max(L(:))).For each labeled component, it calculates the area of that component by summing all the binary values (1s) in the component. This is done by setting component to a binary image where only the pixels of the current label are set to 1.The area of the component is compared to the specified minArea. If the area is greater than or equal to minArea, the label of the component is added to the keepLabels list, indicating that it should be retained.Finally, a binary image BW_cleaned is created using ismember(L, keepLabels). This creates a binary image where only the connected components with labels in the keepLabels list are retained, and others are set to 0.

```
function BW_cleaned = myremovecom(BW, minArea)
    [L, ~] = mysegmentation(BW);
    keepLabels = [];

    for label = 1:max(L(:))
        component = (L == label);
        componentArea = sum(component(:));

        if componentArea >= minArea
            keepLabels = [keepLabels, label];
        end
    end
    BW_cleaned = ismember(L, keepLabels);
end
```

We use the function in the main code to remove the image's noise.The final image, **picture2**, should contain only the main objects of interest with the background removed.



6. The mysegmentation function initializes a label matrix L with the same dimensions as the input binary image BW. It also initializes a variable label to 0, which will be used to assign unique labels to connected components. The function iterates through each pixel of the binary image in a nested loop, examining each pixel one by one. For each pixel, if it is part of a connected component (i.e., its value is 1 in the binary image), the function looks at the neighboring pixels to check if they are labeled with existing labels. It creates a neighbors matrix by extracting the neighboring pixels, and then extracts unique non-zero labels from the neighbors. If there are no neighboring labels (i.e., the current pixel is not part of any existing connected component), a new label is assigned, and label is incremented by 1. This assigns a unique label to this new connected component. If there are neighboring labels, the current pixel is assigned the smallest neighboring label. This is because it's part of the same connected component as its neighbors. The function continues this process for all pixels in the binary image, effectively labeling all connected components. Finally, the

function counts the number of connected components by checking the value of the label variable, and this count is returned as Ne.

```
function [L, Ne] = mysegmentation(BW)
    L = zeros(size(BW));
    label = 0;
    for row = 1:size(BW, 1)
        for col = 1:size(BW, 2)
            if BW(row, col) == 1
                neighbors = [L(max(row-1,1):min(row+1,end), max(col-1,1):min(col+1,end))];
                neighbor_labels = unique(nonzeros(neighbors));
                if isempty(neighbor_labels)
                    label = label + 1;
                    L(row, col) = label;
                else
                    L(row, col) = min(neighbor_labels);
                end
            end
        end
    end
    Ne = label;
end
```

```
figure
[L,Ne]=mysegmentation(picture2);
subplot(1,2,1)
imshow(picture2)

propied=regionprops(L,'BoundingBox');
hold on
for n=1:size(propied,1)
    rectangle('Position',propied(n).BoundingBox,'EdgeColor','g','LineWidth',2)
end
hold off
```

7. For decision making we use the code below. Here's an explanation of the code:

   load TRAININGSET: This line loads a training set data, presumably containing templates or samples of characters for character recognition.

   totalLetters = size (TRAIN, 2): This line calculates the total number of characters in the training set by getting the number of columns in the cell array TRAIN.

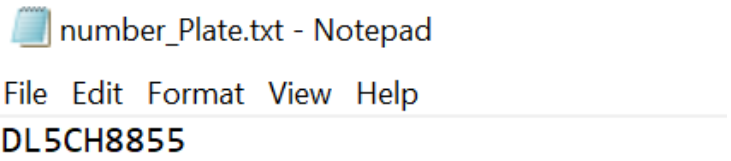
   final_output = [] and t = []: These lines initialize two empty arrays, final_output and t, which will be used to store the recognized characters and intermediate results.

   The code enters a loop (for n = 1: Ne) to process each connected component (likely representing a character) in the picture2 image. Inside the loop, the code identifies the row and column indices of the pixels belonging to the current connected component n using find(L == n).It then extracts the region of interest (ROI) from picture2 based on the identified row and column indices. The ROI (Y) corresponds to the isolated character that needs to be recognized. The code resizes the extracted character (Y) to a common size of 42x24 pixels using imresize. It initializes an array “ro” to store correlation values for each character in the training set. For each character in the training set, the code calculates the correlation between the extracted character Y and the template character using corr2 (correlation coefficient). This step aims to find the best match in the training set for the extracted character. The code finds the character in the training set that has the highest correlation (MAXRO) with the extracted character Y, as well as the position (pos) of that character in the training set. If the highest correlation (MAXRO) is above a certain threshold (0.30 in this case), it is considered a valid recognition, and the recognized character is obtained from the training set (TRAIN {2, pos}). The recognized character is then added to the final_output array. The loop continues to process the next connected component (character) in the image. After the loop has processed all the connected components, the final_output array should contain the recognized characters from the image.

```
load TRAININGSET;
totalLetters=size(TRAIN,2);

final_output=[];
t=[];
for n=1:Ne
    [r,c]=find(L==n);
    Y=picture2(min(r):max(r),min(c):max(c));
    Y=imresize(Y,[42,24]);
    pause(0.2)


    ro=zeros(1,totalLetters);
    for k=1:totalLetters
        ro(k)=corr2(TRAIN{1,k},Y);
    end

    [MAXRO,pos]=max(ro);
    if MAXRO>.30
        out=cell2mat(TRAIN(2,pos));
        final_output=[final_output out];
    end
end
```

8. The result of running this code is that the recognized license plate characters are saved in a text file, and the file is automatically opened for viewing in a text editor or viewer. The text file will contain one recognized character per line.

```
% Printing the plate
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
file = fopen('number_Plate.txt', 'wt');
fprintf(file,'%s\n',final_output);
fclose(file);
winopen('number_Plate.txt')
```



The result is just as we anticipated.

- ***Part 2:***

The code of this part doesn't have much difference from part one, except that we can use built-in functions instead of the custom ones.

This code can be summarized as follows:

1. **Image Input and Preprocessing:**
    - The code begins by allowing the user to select an image file (commonly a license plate image).
    - The selected image is displayed and then resized to a consistent size of 300x500 pixels.
    - The image is converted to grayscale using rgb2gray function.
    - Thresholding is applied to convert the grayscale image to a binary image using imbinarize function, segmenting the characters from the background.

- By using bwareaopen, small connected components and background elements are removed from the binary image.

2. **Connected Component Labeling and Visualization:**
   - The connected components in the binary image are labeled using bwlabel.
   - The bounding boxes of the labeled components are outlined with green rectangles, visualizing the segmentation process.
3. **Character Recognition:**
   - A training set of Persian character templates is loaded.
   - The code iterates through the labeled connected components and extracts regions of interest (ROIs) corresponding to isolated characters.
   - Each ROI is resized to a consistent size (100x80 pixels).
   - For each character in the training set, the code calculates the correlation with the extracted character ROI. If the correlation is above a certain threshold (0.45), it recognizes the character.
   - Recognized characters are accumulated in the final_output array.
4. **Printing and Display:**
   - The recognized characters are written to a text file named 'number_Plate.txt'.
   - The file is opened using the default text file viewer.

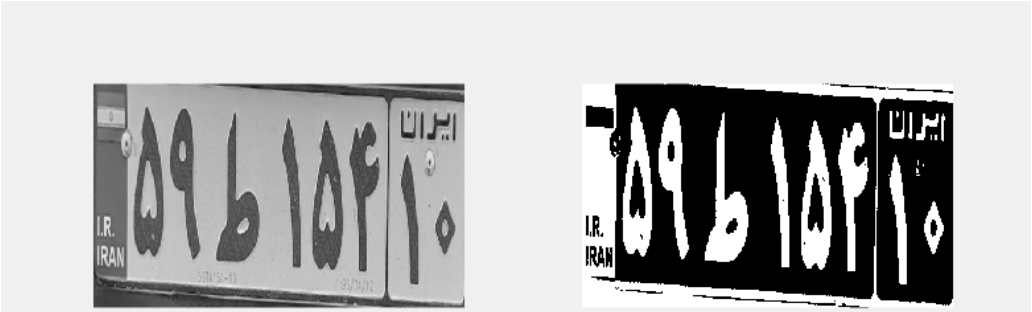
```
% SELECTING THE TEST DATA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[file,path]=uigetfile({'*.jpg;*.bmp;*.png;*.tif'},'Choose an image');
s=[path,file];
picture=imread(s);
figure
subplot(1,2,1)
imshow(picture)
picture=imresize(picture,[300 500]);
subplot(1,2,2)
imshow(picture)

%RGB2GRAY
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
picture=rgb2gray(picture);
figure
subplot(1,2,1)
imshow(picture)

% THRESHOLDIG and CONVERSION TO A BINARY IMAGE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
threshold = graythresh(picture);
picture =~imbinarize(picture,threshold);
```

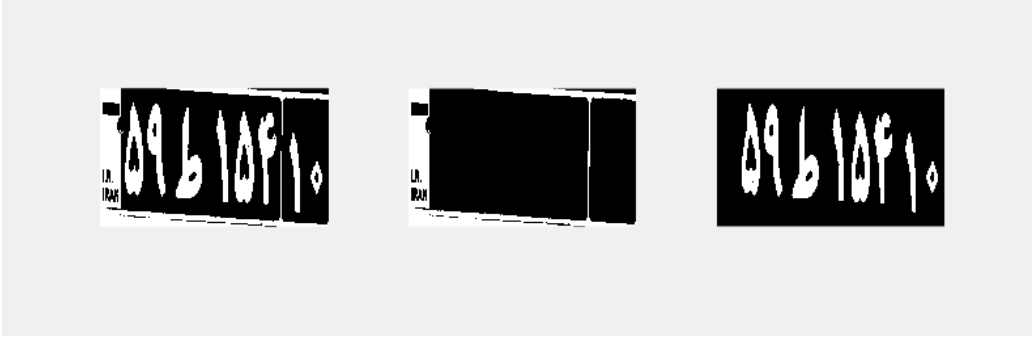
```
picture =~imbinarize(picture,threshold);
subplot(1,2,2)
imshow(picture)

% Removing the small objects and background
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure
picture = bwareaopen(picture,800); % removes all connected components (objects) that have fewer than 800 pixels from the binary image
subplot(1,3,1)
imshow(picture)
background=bwareaopen(picture,7000);
subplot(1,3,2)
imshow(background)
picture2=picture-background;
subplot(1,3,3)
imshow(picture2)
```

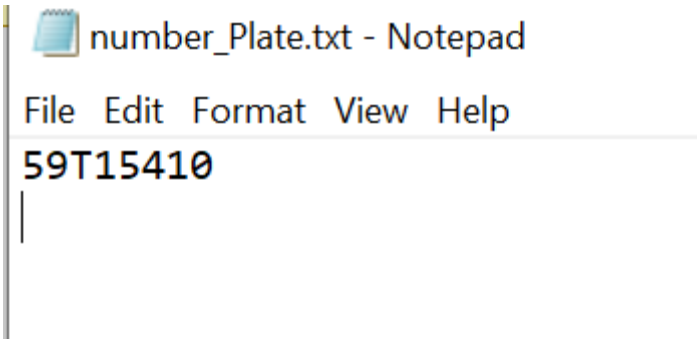
```
%%
% Labeling connected components
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure
imshow(picture2)
[L,Ne]=bwlabel(picture2);
propied=regionprops(L,'BoundingBox');
hold on
for n=1:size(propied,1)
    rectangle('Position',propied(n).BoundingBox,'EdgeColor','g','LineWidth',2)
end
hold off
```

```
% Decision Making
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Loading the mapset
load TRAININGSET;
totalLetters=size(TRAIN,2);


figure
final_output=[];
t=[];
for n=1:Ne
    [r,c]=find(L==n);
    Y=picture2(min(r):max(r),min(c):max(c));
    imshow(Y)
    Y=imresize(Y,[100,80]);
    imshow(Y)
    pause(0.2)
    ro=zeros(1,totalLetters);
    for k=1:totalLetters
        ro(k)=corr2(TRAIN{1,k},Y);
    end
%%
    [MAXRO,pos]=max(ro);
    if MAXRO>.45
        out=cell2mat(TRAIN(2,pos));
        final_output=[final_output out];
    end
end



% Printing the plate
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
file = fopen('number_Plate.txt', 'wt');
fprintf(file,'%s\n',final_output);
fclose(file);
winopen('number_Plate.txt')
toc
```

Here are the results for an example:







The outcome matches what we were hoping for.

- ***Part 3:***

For extracting the license plate region, our idea is to find parts of the image where there is a significant difference in color, both in terms of the number and intensity of color changes.

We process an image represented by the variable "picture" to extract a region of interest (ROI) based on certain criteria for changes in pixel values in both the vertical and horizontal directions. Here's an explanation of the code:

1. **Initialization:**

   y_down is initially set to width - 100, which is defining the lower y-coordinate boundary.

   y_top is set to 100, presumably defining the upper y-coordinate boundary.

   x_right is initially set to length - 100, representing the right x-coordinate boundary.

   x_left is set to 100, representing the left x-coordinate boundary.

   colChangeLimit, colMaxChanges, and colsChangesCount are initialized to 0, 1, and an array of zeros, respectively.

2. **Column Changes Calculation:**

   Two nested loops iterate over the columns (indexed by j) and rows (indexed by i) of an image represented by the variable picture.

   In the inner loop, it counts how many times the pixel values change from one row to the next (indicating changes in the vertical direction) in a specific column.

   The count is stored in the ChangesCount variable for each column.

   The maximum count of changes (colChangeLimit) and the corresponding column index (colMaxChanges) are updated if certain conditions are met.

3. **Column Boundary Adjustment:**

   After finding the maximum change in column count, the code adjusts the x_left boundary by searching for a suitable point where the column changes match the colChangeLimit with some tolerance.

4. **Row Changes Calculation:**

   A similar process as in step 2 is repeated, but this time, the code counts changes in pixel values from one column to the next (indicating changes in the horizontal direction) in each row of the image.

5. **Row Boundary Adjustment:**

   After finding the maximum change in row count, the code adjusts the y_top and y_down boundaries by searching for suitable points where the row changes match the RowChangeLimit with some tolerance.

6. **Calculating yChange:**

   The code calculates the difference between y_down and y_top and checks if it's less than 60. If it is, it adjusts y_down to make sure it's at least 60 pixels below y_top.

7. **Extracting the Region of Interest (ROI):**

   Finally, a portion of the input image is extracted, specifically the area defined by y_top, y_down, x_left, and x_right.

The remaining portion of the code remains unchanged, just as described in part 2.

```
% Extract the license plate region
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
y_down = width - 100;
y_top = 100;

x_right = length - 100;
x_left = 100;

colChangeLimit = 0;
colMaxChanges = 1;
colsChangesCount = zeros(1, length);

for j=1: length
    ChangesCount = 0;
    for i=1: width - 1
        if picture(i + 1, j) ~= picture(i, j)
            ChangesCount = ChangesCount + 1;
        end
    end
```

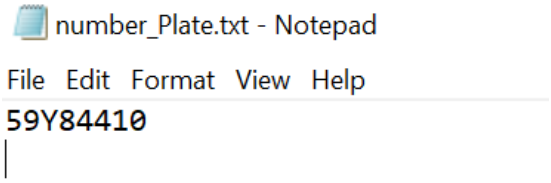
```
    colsChangesCount(i) = ChangesCount;
    if ChangesCount > colChangeLimit && j > 300 && j < 400
        colChangeLimit = ChangesCount;
        colMaxChanges = j;
    end
end

for j=220: colMaxChanges
    if abs(colChangeLimit - colsChangesCount(j)) < 30 && colMaxChanges - j < 230
        x_left = j;
        break;
    end
end

for j=length - 200:-1: colMaxChanges
    if abs(colChangeLimit - colsChangesCount(j)) < 30 && j - colMaxChanges < 300
        x_right = j;
        break;
    end
end
```

```
RowChangeLimit = 0;
rowMaxChanges = 1;
rowsChangesCount = zeros(1, width);

for i=1: width
    ChangesCount = 0;
    for j=1: length - 1
        if picture(i, j + 1) ~= picture(i, j)
            ChangesCount = ChangesCount + 1;
        end
    end
    rowsChangesCount(i) = ChangesCount;
    if ChangesCount > RowChangeLimit && i > 340 && i < 500
        RowChangeLimit = ChangesCount;
        rowMaxChanges = i;
    end
end
```

```
for i=100: rowMaxChanges
    if abs(RowChangeLimit - rowsChangesCount(i)) < 20 && rowMaxChanges - i < 50
        y_top = i;
        break;
    end
end

for i=width - 100:-1: rowMaxChanges
    if abs(RowChangeLimit - rowsChangesCount(i)) < 20 && i - rowMaxChanges < 50
        y_down = i;
        break;
    end
end

yChange = y_down - y_top;

if yChange < 60
    y_down = y_down + 80 - yChange;
end
```

```
plate = picture(y_top:y_down,x_left:x_right);
imshow(plate);
```

Here are the results for an example:



The result aligns with our desired expectations.