Politecnico di Bari

IMAGE PROCESSING COURSE

PROJECT:
OPTIC DISC AND MACULA RECOGNITION IN RETINAL IMAGES

STUDENT: KLAJDI ISTERI
DATE: 2/12/2021

Summary

Introduction

Overview

The main purpose of this project is the analysis of retina's images and the creation of a recognition algorithm based on image processing techniques for the identification of optic disc and macula.

We will start by setting up the development environment then we will focus on the exploration of the data contained in the dataset of images and develop the algorithm for the image's segmentation.

Finally, we'll test the algorithm on the test dataset to analyze the results and verify the accuracy.

Enviroment Setup

MATLAB

For this project we will work on Windows Environment using MATLAB.

MATLAB (an abbreviation of "MATrix LABoratory") is a proprietary multi-paradigm programming language and numeric computing environment developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms.

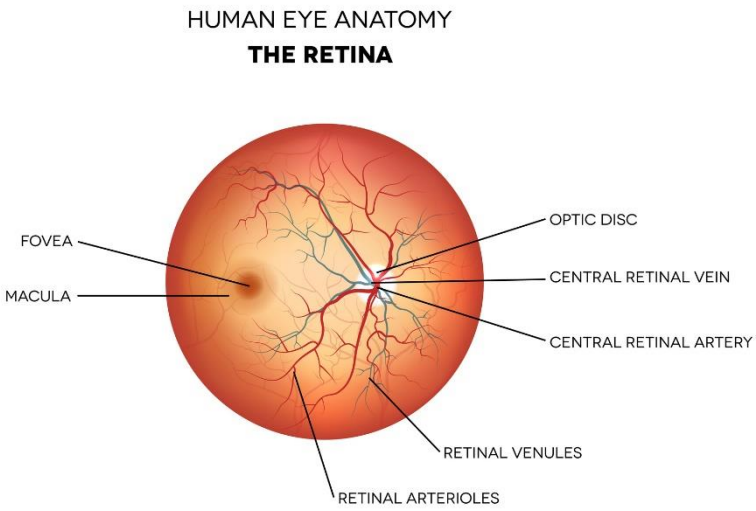
Source: https://it.mathworks.com/products/image.html

Although MATLAB is intended primarily for numeric computing, the use of the Image Processing Toolbox™ [1] provides a comprehensive set of reference-standard algorithms and workflow apps for image processing, analysis, visualization, and algorithm development. With this toolbox we can perform image segmentation, image enhancement, noise reduction, geometric transformations, and image registration using deep learning and traditional image processing techniques. The toolbox supports processing of 2D, 3D, and arbitrarily large images.

Image Processing Toolbox apps let us automate common image processing workflows. We can interactively segment image data, compare techniques, and batch-process large datasets. Visualization functions and apps let us explore images, adjust contrast, create histograms and perform different processing approaches.

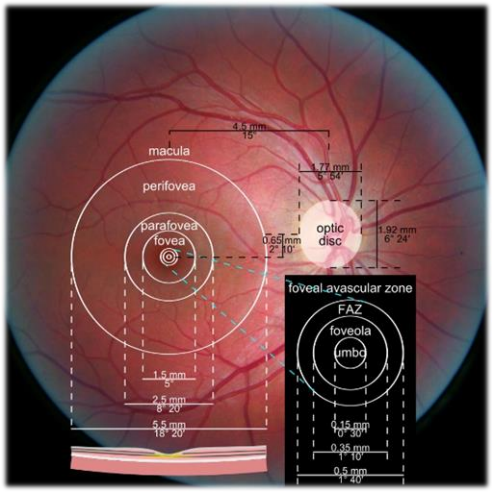
Data Exploration and Visualization

In the image below we can see the structure of the retinal image, as we can see the optic disc and the macula are respectively the brightest and the darkest parts of the image.



Source: https://www.advancedretinaassociates.com/patient-education/human-eye-anatomy-retina

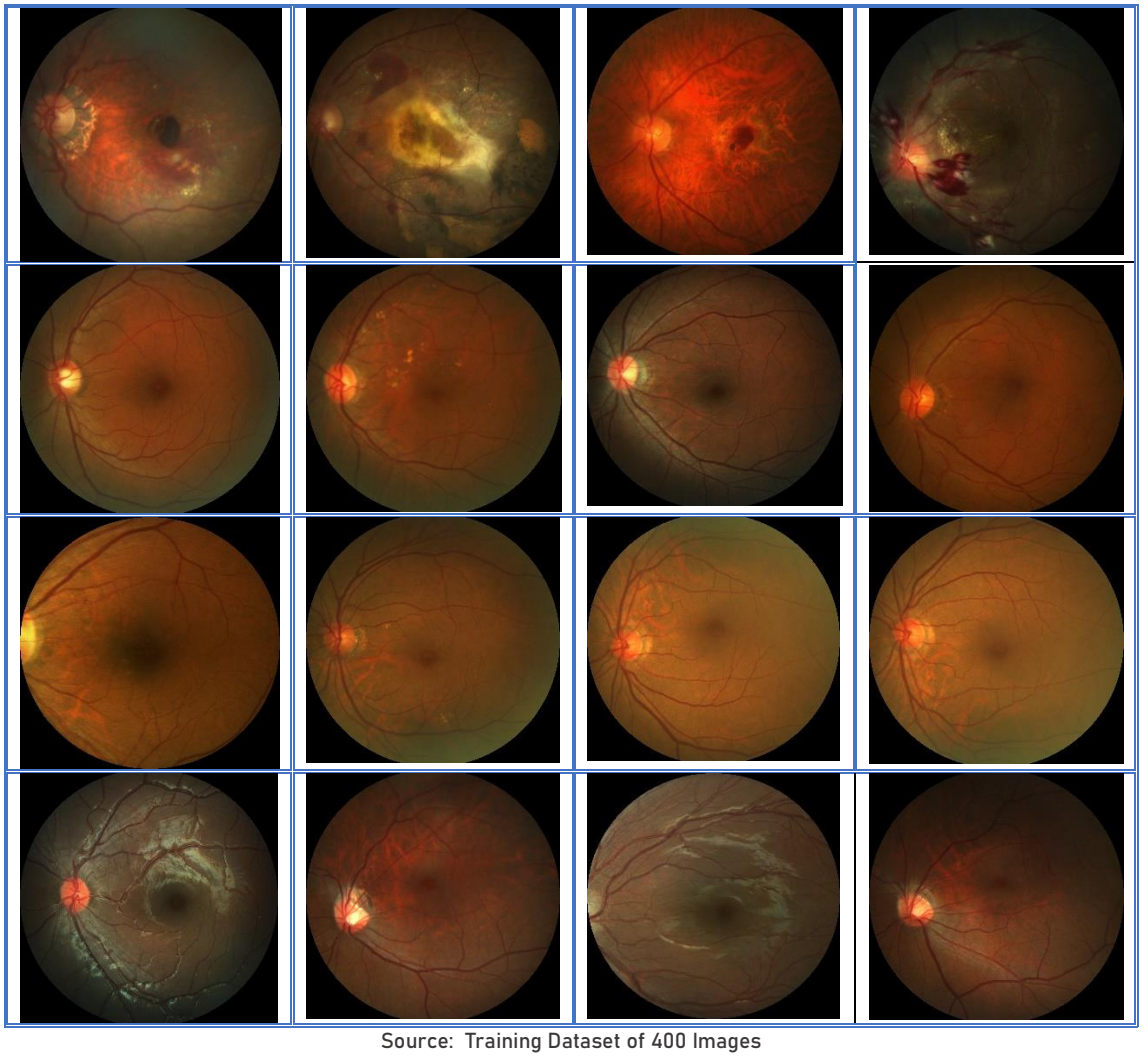
In the image below we can also see that usually the macula's diameter is 2.86 times the diameter of the optic disc, this will help us later when we will draw the detection circles in the images.



Source: https://gl.m.wikipedia.org/wiki/Ficheiro:Macula.svg

Exploration

To better understand the problem and the creation of the algorithm we will explore data to find relations between images.



Source: Training Dataset of 400 Images

As we can see in the image above, we have RGB images with different color channels distributions, so doing a color thresholding technique to segment the images would be a great approach, but in this color space it will be quite difficult and will also give us 3 different variables to work with so, we need to change approach to create a better optimized algorithm.

We have seen before that a better strategy is to point the darkest and brightest parts of the image and for doing so, we will perform a color space transformation from RGB to YCBCR.

In standard computer graphics RGB (8-bit per channel) encoding, each pixel's intensity value ranges from 0 to 255. (0,0,0) is black and (255,255,255) is white.

In YCBCR (*always* 8-bit per channel) encoding [2]:

- Y ranges from 16 to 235. 16 is the darkest, 235 is the brightest.
- CB and CR range from 16 to 240. 128 is the zero-point.

Thus, (16, 128, 128) is pure black, and (235, 128, 128) is pure white. The following formulae can be used to transform pixels between RGB and YCBCR representations.

Here we use lower-case letters to specify components with normalized scaling: r', g', b', y': [0.0, 1.0]; and cB, cR: [-0.5, +0.5].

Conversion to $y'c_Bc_R$ from r'g'b'			
$y' =$	0.299 * r' +	0.587 * g' +	0.114 * b'
$c_B =$	-0.168736 * r' +	-0.331264 * g' +	0.500 * b'
$c_R =$	0.500 * r' +	-0.418688 * g' +	-0.081312 * b'

Conversion to r'g'b' from $y'c_Bc_R$			
r' =	1.0 * y' +	0 * c_B +	1.402 * c_R
g' =	1.0 * y' +	-0.344136 * c_B +	-0.714136 * c_R
b' =	1.0 * y' +	1.772 * c_B +	0 * c_R

r' = R' / 255 g' = G' / 255 b' = B' / 255	Y' = (y' * 219) + 16 C_B = (c_B * 224) + 128 C_R = (c_R * 224) + 128

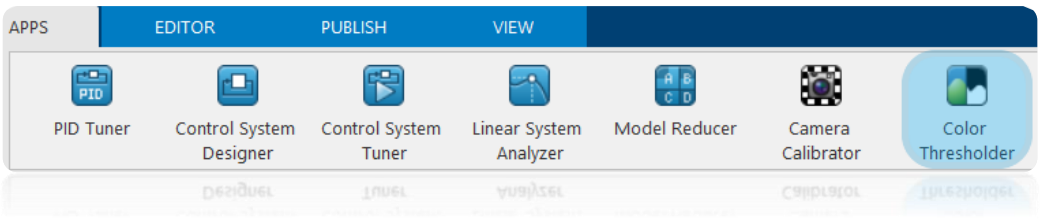
Source: https://www.mir.com/DMG/ycbcr.html

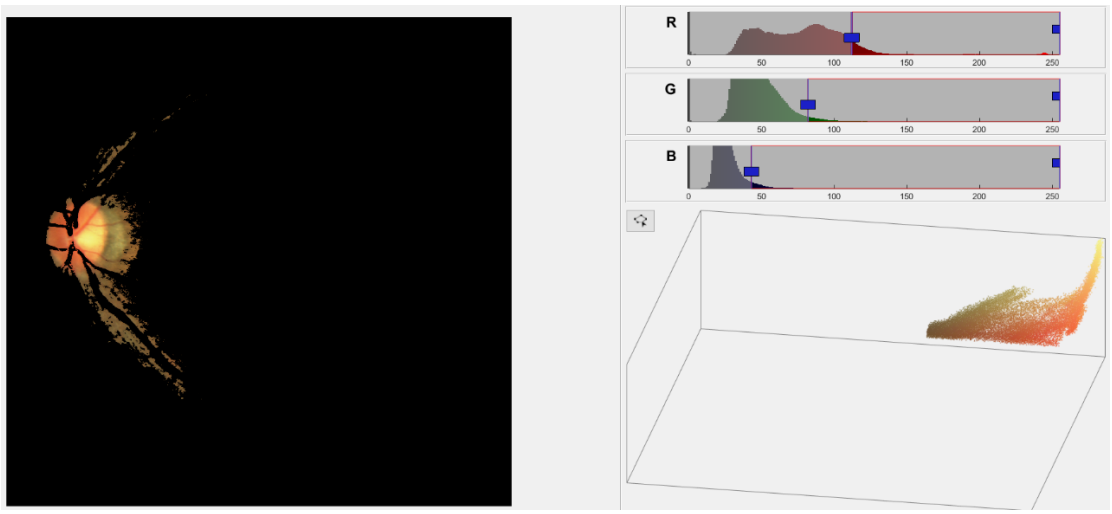
We need to scale 8-bit R', G', and B' values by a factor of 1/255 to use these equations. Likewise, proper 8-bit Y' needs to be scaled by 219 and offset by 16. Proper 8-bit CB and CR need to be scaled by 224 and offset by 128.

The Y (luminance) parameter is the one of our interest because helps us thresholding the image by its brightness.

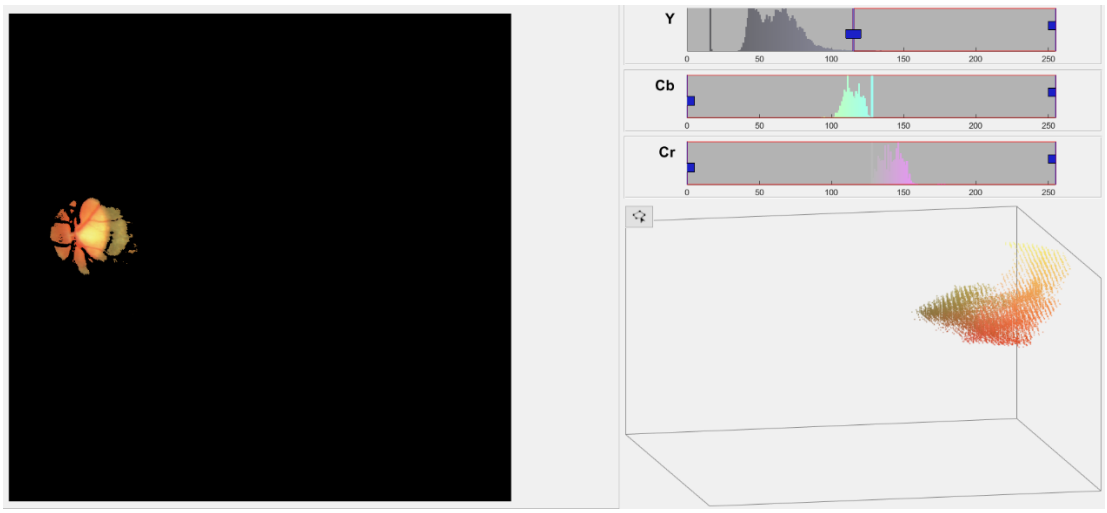
Visualization

In MATLAB's Image Processing Toolbox, we have a useful app to show the image color thresholding, we will use this tool just to show what are the result but in the actual code we will automate all these processes by using a trained algorithm, for now let's see the results in RGB and YCBCR color spaces:

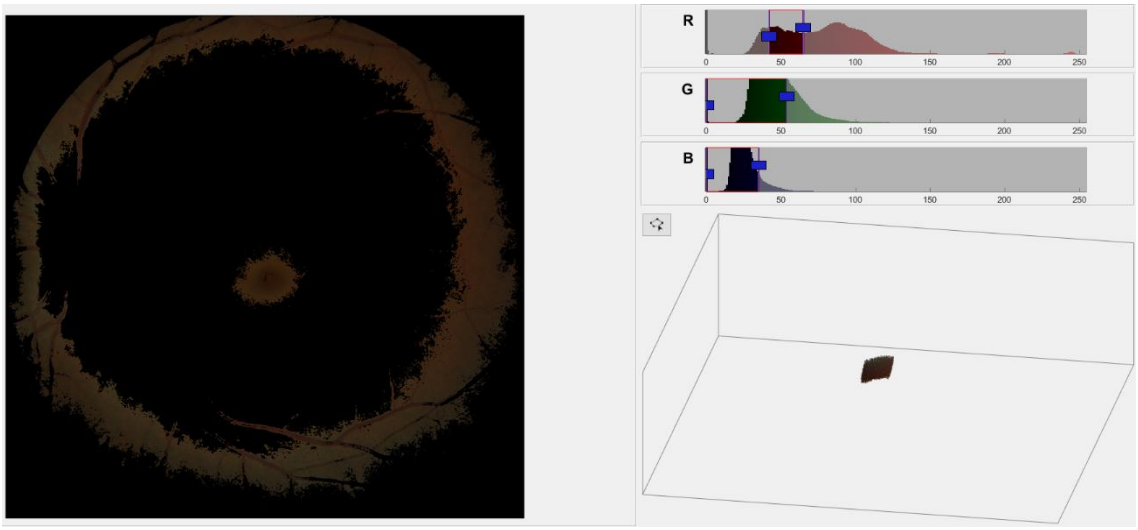




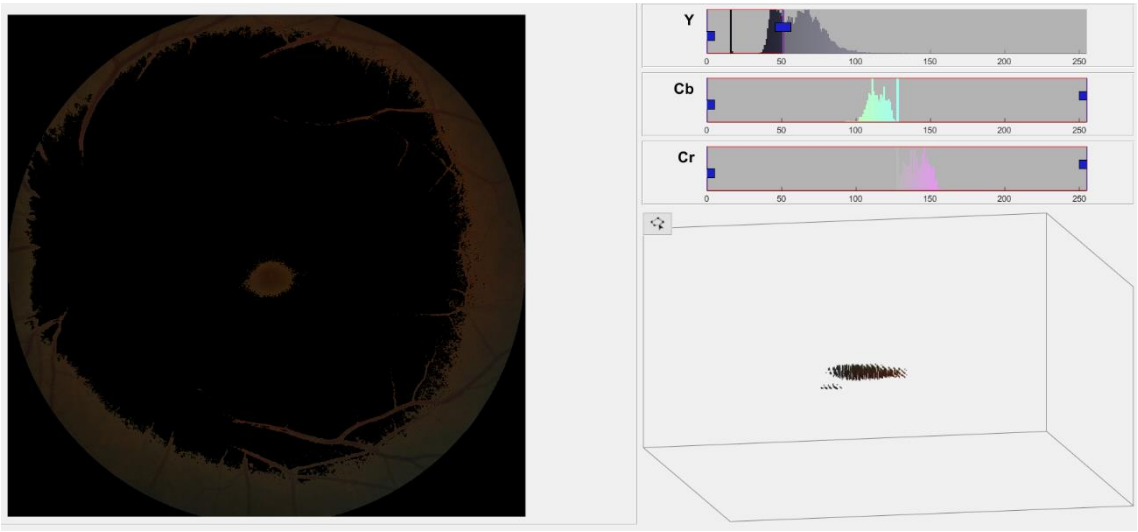
Color thresholding for optic disc recognition in RGB's color space by use of three channels, R, G and B.



Color thresholding for optic disc recognition in YCBCR's color space by use of one channel (Y).



Color thresholding for macula's recognition in RGB's color space by use of three channels, R, G and B.



Color thresholding for macula's recognition in YCBCR's color space by use of one channel (Y).

Has we can see in the above images using the YCBCR color space we are not only saving time (less variables to manage) but we are also getting a better precision.

Recognition Algorithm

We need to automate the thresholding process, for doing so we are using an approach similar to a Machine Learning algorithm, but we are doing it from scratch.

MATLAB's code is divided in two parts:

1. Training Algorithm
2. Main Algorithm

Training Algorithm for Optic Disc

This function does a dynamic color segmentation of the images contained in the Training Dataset.

The function tries the Y parameter over all the images and creates a cost function based on the values of the roundness of the region that represents the optic disc, also some constraints and image processing techniques were applied to increase the accuracy.

Let's see step by step what this code does:

```
function [cost] = optic_disc_identifier(Y)
  cost = 0;

  imds = imageDatastore("Training","FileExtensions",[".jpg",".tif"]);
  numImages = numel(imds.Files);

  for i=1:numImages...
end
```

The optic_disc_identifier function takes Y parameter as Input and gives a cost as Output, this procedure helps us for the training part because we can simply use a MATLAB's minimizer function that tries different values of Y to get the best value that minimizes the cost of the identifier.

```
options = optimset('Display','iter', 'TolX', 1);
x = fminbnd(@optic_disc_identifier,x1,x2, options);
disp(x);
```

“fminbnd” is a one-dimensional minimizer that finds a minimum for a problem specified by:

$$\min_x f(x) \text{ such that } x_1 < x < x_2.$$

x, x1, and x2 are finite scalars, and f(x) is a function that returns a scalar.

“x = fminbnd(f(x), x1, x2)” returns a value x that is a local minimizer of the scalar valued function that is described in “fun” in the interval x1 < x < x2.

The initial cost is set to zero and will increase till the total cost is reached in the end of the for loop, basically when all the images were analyzed.

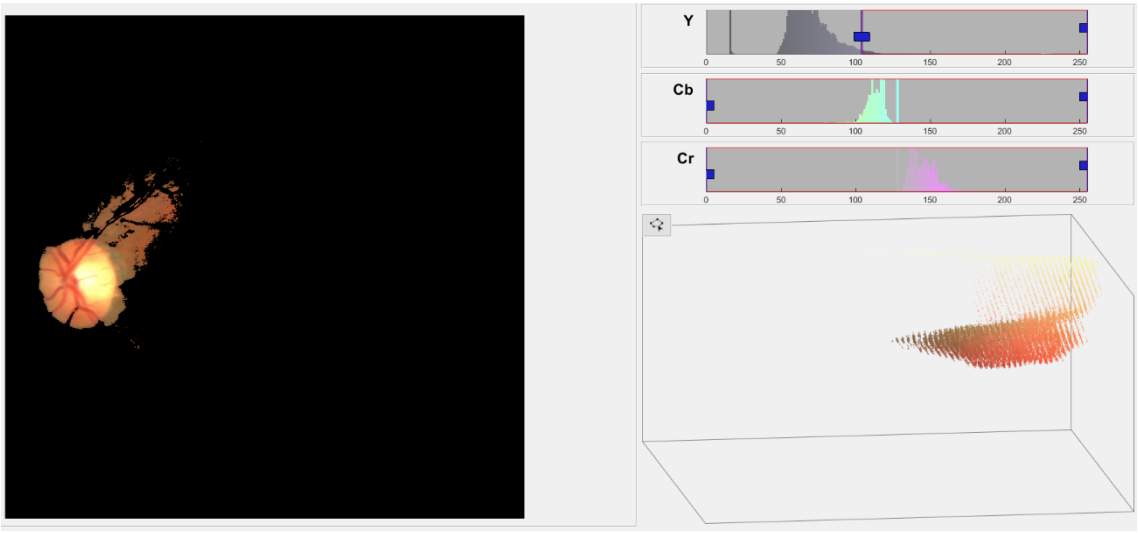
Then at each call of the minimizer, so when a new value of “x” is tried, the total cost resets to zero and is re-calculated to be compared to the others to find the minimum cost which is associated to the “x” we are searching.

Now, let’s see what happens in the for loop for all the images in the dataset:

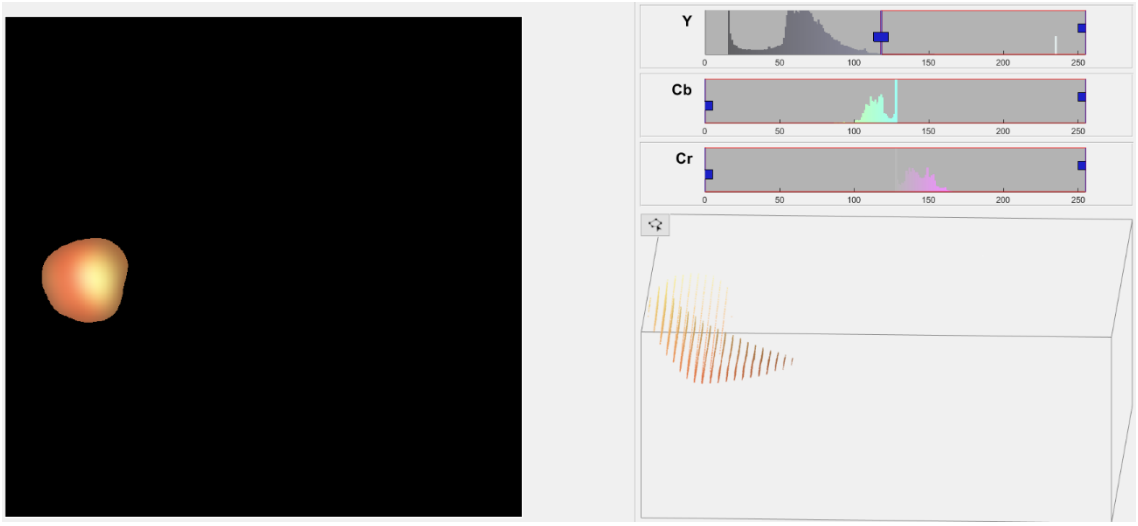
```
%Create a gaussian filter and use it to blur the image
%(Pre-processing)
alpha = 4;
sigma=10*alpha;

kernel = fspecial('gaussian',4*sigma+1,sigma);
imgT = imfilter(img,kernel,'symmetric');
```

Basically, here we are creating a gaussian kernel that is applied to the image [3], applying a gaussian blur has the effect of reducing the image's high-frequency components, a gaussian blur is thus a low pass filter used to ensure that spurious high-frequency information does not appear in the down sampled image.

Gaussian blurs have nice properties, such as having no sharp edges, and thus do not introduce ringing into the filtered image so applying it helps us in the thresholding segmentation, let's see it:



Color thresholding for optic disc whitout Gaussian Filter.



Color thresholding for optic disc after Gaussian Filter is applied.

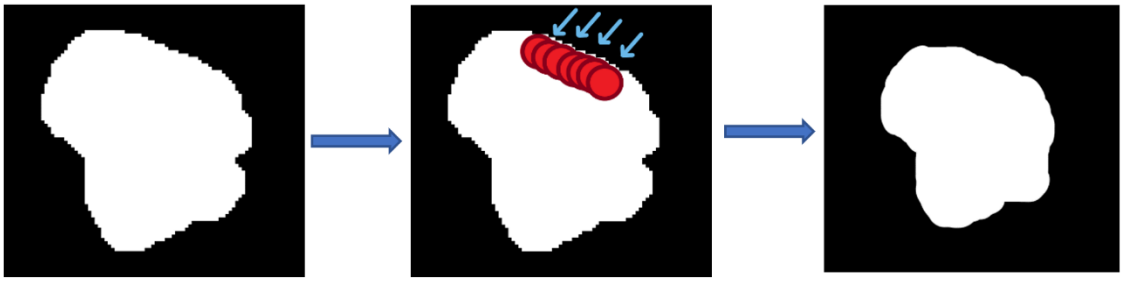
This transformation not only gives us more accuracy (see homogeneity) but gives us also a more rounded edge and this property will help us later.

```
%Color tresholding based on Y value
YD = (YCbCrD(:,:,1) >= 100 + Y ) & (YCbCrD(:,:,1) <= 250);

%Performs a morphological opening doing an erosion followed by a
%dilation
se = strel('disk',20);
YD = imopen(YD,se);
```

In this part we are doing the thresholding, we start from a fixed valued obtained after a brief analysis of the images, usually after 100 we start isolating the optic disk because of the pixels luminance intensity and this also speed up our minimizer which must try less values to find the minimum. MATLAB gives us a max value of 250 of the Y but we know that the real max value after the transformation to YCBCR is 235, we leave it to MATLAB's max as it is.

NOTE: In MATLAB this operation creates a mask, so a logical array composed of 1 and 0s.

After we do a morphological image opening [4], the effect of opening can be quite easily visualized. Imagine taking the structuring element "strel" and sliding it around inside each foreground region, without changing its orientation. All pixels which can be covered by the structuring element with the structuring element being entirely within the foreground region will be preserved. However, all foreground pixels which cannot be reached by the structuring element without parts of it moving out of the foreground region will be eroded away. After the opening has been carried out, the new boundaries of foreground regions will all be such that the structuring element fits inside them.

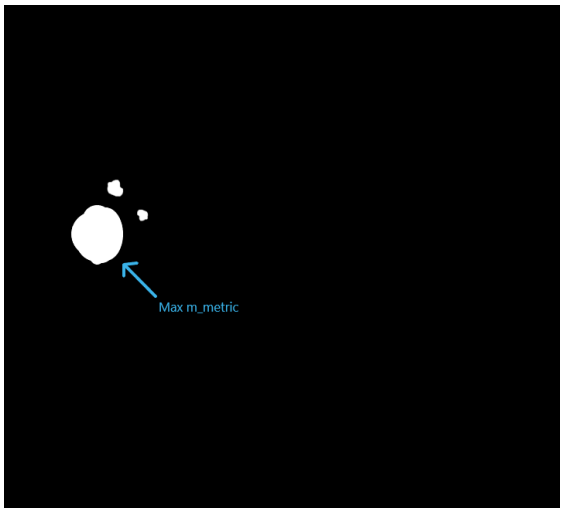


Morphological opening whith disk structuring element.

```
%Takes boundaries of image
[bound,L] = bwboundaries(YD,'noholes');
stats = regionprops(L,'Area','Centroid');

%Set of max value of roundness between boundaries
m_metric = 0;

    %Loop over the boundaries
    for k = 1:length(bound)

      %Obtain (X,Y) boundary coordinates corresponding to label 'k'
      boundary = bound{k};

      %Compute a simple estimate of the object's perimeter
      delta_sq = diff(boundary).^2;
      perimeter = sum(sqrt(sum(delta_sq,2)));

      %Obtain the area calculation corresponding to label 'k'
      area = stats(k).Area;

      %Compute the roundness metric
      metric = 4*pi*area/perimeter^2;

      %Find max value of roundness in boundaries
      if metric > m_metric
          m_metric = metric;
      end

    end
```

In this part of the code, we take all the boundaries and loop over all of them to find the boundary with the max value of roundness. We initially set the m_metric (which is the max roundness) equal to zero so that the first metric found is the new max and the next metric to be found are confronted to get region with the maximum value of roundness.

This loop is done because on some occasions we can find more boundaries in the images, and we want to take the best one which is the optic disk due to its form.




```
%Constraint on shape
if(m_metric < 0.80)
    m_metric = 0.00;
end

%Sum of all costs
cost = cost + (1-m_metric);
disp('Roundness: ');
disp(m_metric);
```

Here we are putting a constraint on the max value of roundness that we want, basically we are saying that we want the most circular object, in this function we have that:

$$0 \leq (\text{m\_metric} = \text{max roundness}) \leq 1$$

Where: Min = 0 And Max = 1 = Perfect Circle

Here we see that at each iteration, so at every new image, we sum the new cost to get the total cost but we can also see that we are taking "1 - m_metric" this because the bigger the value of roundness and the bigger is the m_metric so that we have a lower value of the cost and a lower value of the total cost to be minimized.

So, summing all up, we created an algorithm which thresholds the images to segment them and to take the best possible Y that minimizes the values of the total cost based on the found roundness of the optical disk, this value of Y will be used in the main algorithm to get the best results in the test dataset.

Training Algorithm for Macula

Basically, the algorithm for macula is like the one used for the optic disc till the morphological opening, after that step we have some different processing:

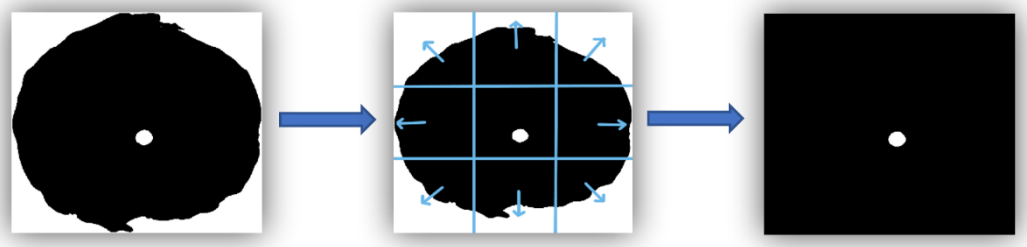
```
%Creates a void matrix to put the future mask
image_thresholded = (size(i2o));

%Loop over all rows and columns
for ii=1:size(i2o,1)
    for jj=1:size(i2o,2)...
end
```

We took the size of the morphologically opened image "i2o" to create a void matrix where we'll put the mask obtained in the for loops that we'll see now:

```
%Get pixel value
pixel=i2o(ii,jj);

  %Makes all non center's pixel equal to zero
  if (jj < round(size(i2o,2)*(1/3)) || jj > round(size(i2o,2)*(2/3))) || (ii < round(size(i2o,1)*(1/3)) || ii > round(size(i2o,1)*(2/3)));
      new_pixel=0;
  else
      new_pixel = pixel;
  end
```

In this part we are moving through each pixel of the image by the two indexes of the for loops, we are checking if the pixel is not in the center of the image, if so, we are setting it to zero.

Let's see it to understand it better:

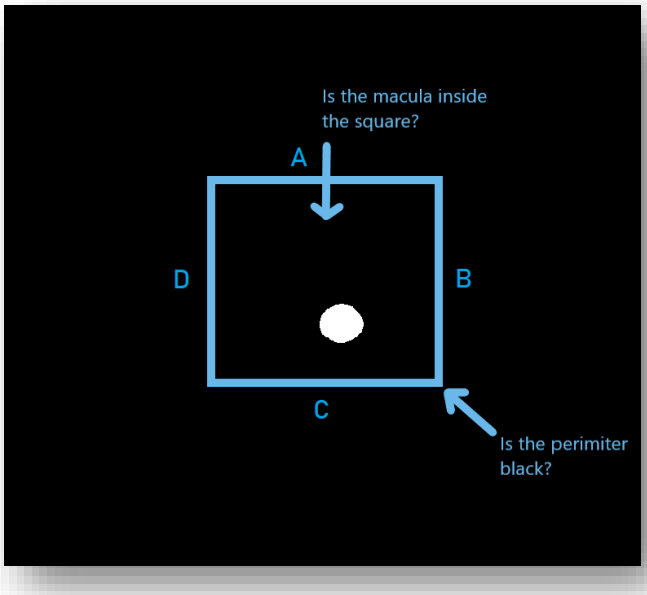


```
%Control of perimeter A,B,C,D (black perimeter)
if (ii == round(size(i2o,1)*(1/3)) && jj >= round(size(i2o,2)*(1/3)) && jj <= round(size(i2o,2)*(2/3)))
    A = A + pixel;
end

if (ii == round(size(i2o,1)*(2/3)) && jj >= round(size(i2o,2)*(1/3)) && jj <= round(size(i2o,2)*(2/3)))
    C = C + pixel;
end

if (jj == round(size(i2o,2)*(1/3)) && ii >= round(size(i2o,1)*(1/3)) && ii <= round(size(i2o,1)*(2/3)))
    D = D + pixel;
end

if (jj == round(size(i2o,2)*(2/3)) && ii >= round(size(i2o,1)*(1/3)) && ii <= round(size(i2o,1)*(2/3)))
    B = B + pixel;
end

%Control if internal of square is not void
if (ii > round(size(i2o,1)*(1/3)) && ii < round(size(i2o,1)*(2/3)) && jj > round(size(i2o,2)*(1/3)) && jj < round(size(i2o,2)*(2/3)))
    square = square + pixel;
end

%Save new pixel value in thresholded image (mask)
image_thresholded(ii,jj)=new_pixel;
```

In this part of the code we are simply checking if the perimeter of the center square is black, to do so we are saving for each side of the square (A,B,C,D) the sum of the values of the pixels, so if the values of these variables are zero, we know that we have a black perimeter.

In the other hand we need to know if in the internal part of the central square we have the macula's mask so we need to know if the value of the sum of all the internal pixels of the square is not zero.

These two parts of code are necessary for the constraints that we are going to apply later to get the best results possible, so the best value of Y.



All the checks of boundaries are the same as for the optic disc, we now apply the constraints:

```
%Control of white image
if(A~=0 && B~=0 && C~=0 && D~=0)
    m_metric = 0.00;
end

%Control of black image
if(square == 0)
    m_metric = 0.00;
end

%Constraint on shape
if(m_metric < 0.80)
    m_metric = 0.00;
end
```

We check if the perimeter of the central square is white, if so that means that all the square is white and we also check if the variable “square” that has the sum of all the pixels inside the perimeter is zero, if so that means that all the image is completely black.

In both the situations we set the value of “m_metric” which is the value of the max roundness, to zero and then we apply the constraint on the admissible value of roundness.

So, summing all up, here we also created an algorithm which thresholds the images to segment them and to take the best possible Y that minimizes the values of the total cost based on the found roundness of the macula, this value of Y will be used in the main algorithm to get the best results in the test dataset.

Main Algorithm

In the main function we threshold the images of the test dataset with the values of the Y found by the minimizers in the training dataset.

The procedures to get the masks are basically the same of the functions minimized, so once we get the masks we simply point at the optic disk and macula in the original RGB image by using the centroid calculated in the masks.

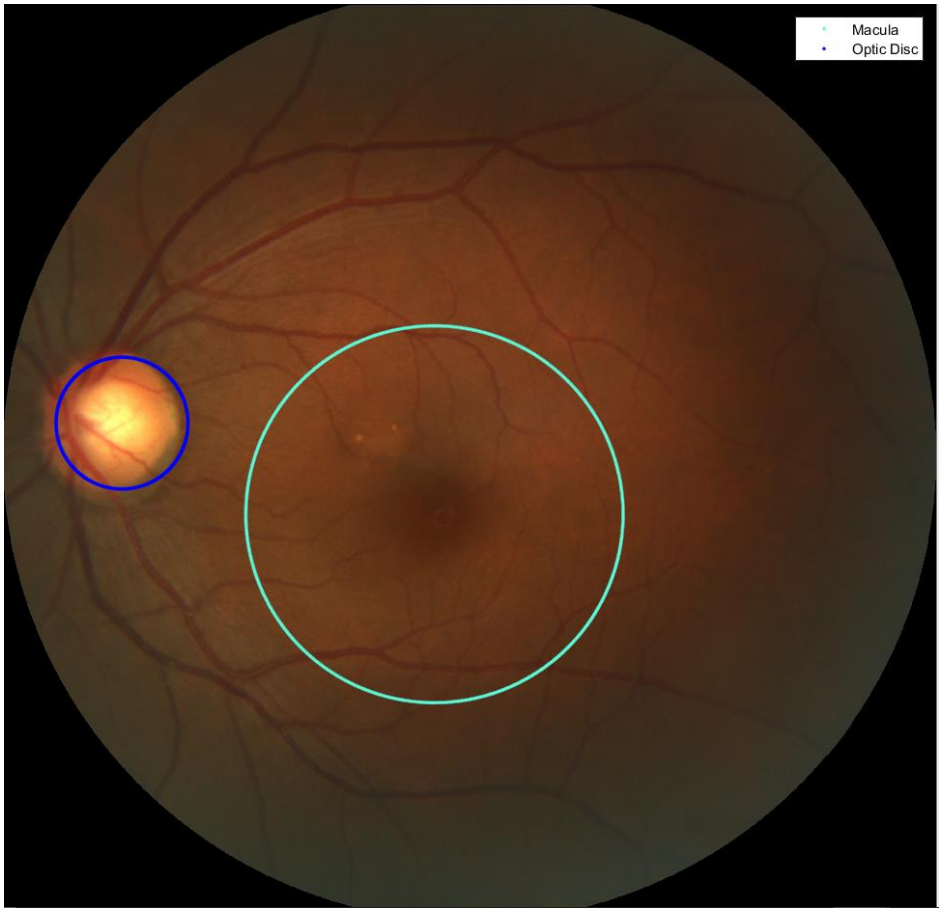
```
%Calculus of centroid for Optic Disc
pod = regionprops(Y,'Centroid');
bool1 = isempty(pod);
if(bool1 == 0)
    rod=150;
    xod = pod(1).Centroid(1);
    yod = pod(1).Centroid(2);
    ang=0:0.01:2*pi;
    xpod=rod*cos(ang);
    ypod=rod*sin(ang);
else
    xod = 0;
    yod = 0;
    xpod=1;
    ypod=1;
end
```

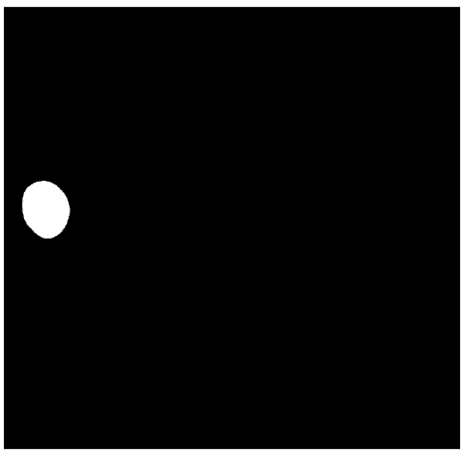
```
%Calculus of centroid for Macula
pm = regionprops(image_thresholded,'Centroid');
bool2 = isempty(pm);
if(bool2 == 0)
    rod=150;
    rm=rod*2.86;
    xm = pm(1).Centroid(1);
    ym = pm(1).Centroid(2);
    ang=0:0.01:2*pi;
    xpm=rm*cos(ang);
    ypm=rm*sin(ang);
else
    xm = 0;
    ym = 0;
    xpm=1;
    ypm=1;
end
```

We check if the centroids aren't void and then we set the values of x and y that we will use to print the circles that identifies the optic this and the macula.

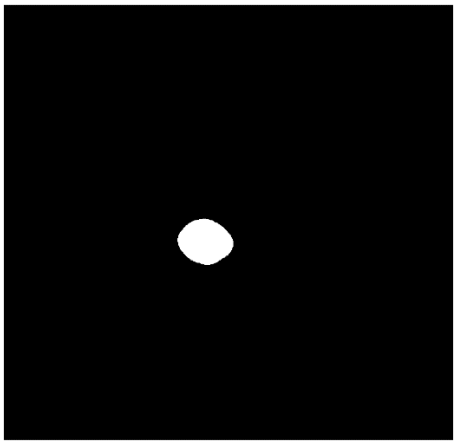
```
%Visualization of detected Macula and Optic Disc
figure(1), imshow(img,'border','tight'), hold on,
scatter(xm+xpm,ym+ypm,'.','MarkerEdgeAlpha',0.1, 'MarkerEdgeColor', '#5afada'),
%since the brightest part of optic disc is in the right part we left shift the centroid
scatter(xod-(xod*0.20)+xpod,yod+ypod,'.','MarkerEdgeAlpha',0.1, 'MarkerEdgeColor', 'blue');
legend('Macula','Optic Disc');
```

Let's see an example of the final expected result:





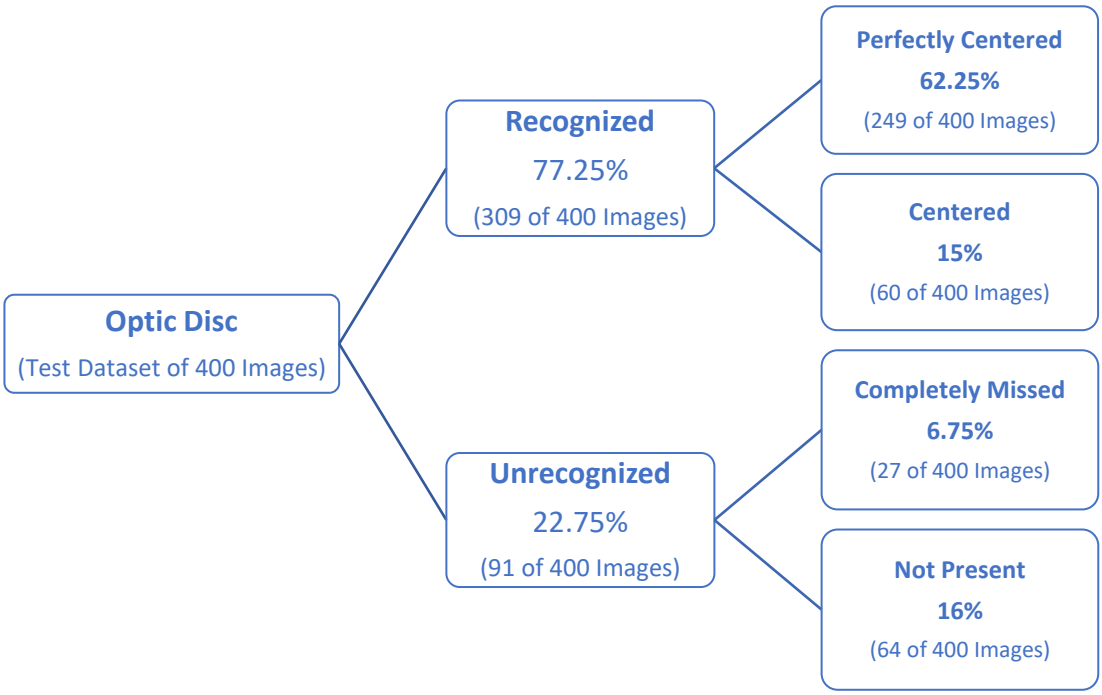
**Optic Disc's Mask*

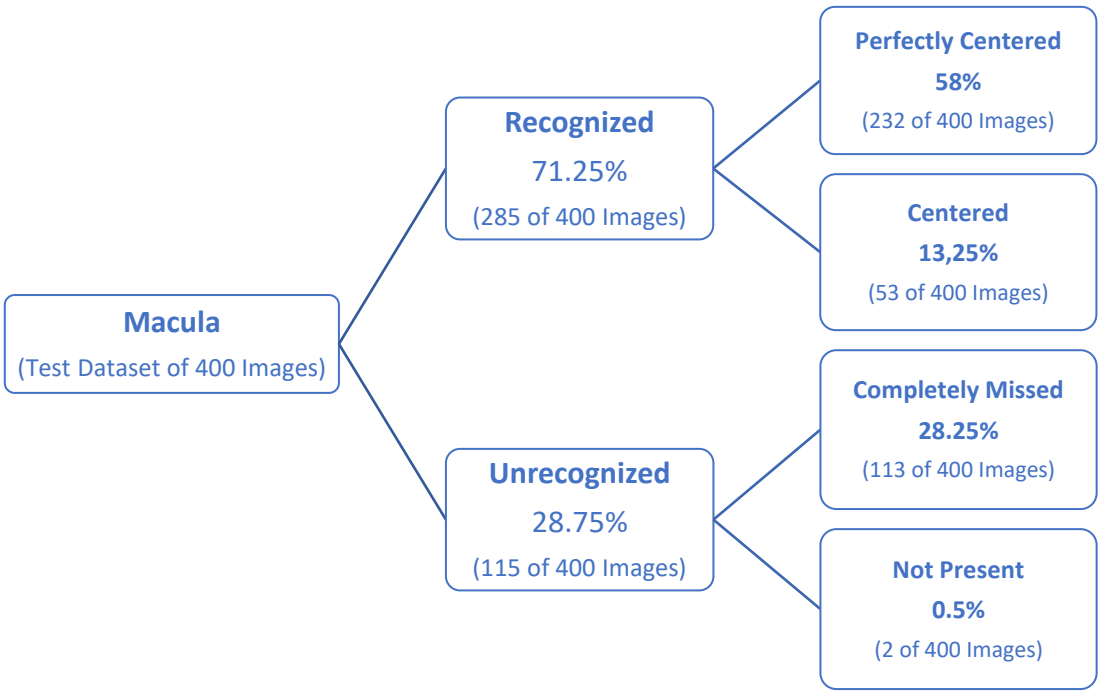


**Macula's Mask*

Conclusion

When we applied the algorithm on the test dataset, we got the following results:





Where:

- Perfectly Centered: *circle aligned with perimeter of optic disc or macula*;
- Centered: *optic disc or macula inside of the circle but not perfectly aligned*;
- Completely Missed: *optic disc or macula completely outside of the detection circle*;
- Not Present: *optic disc or macula are not present in the image*.

Analysis of Results

The algorithm is precise when identifying the optic disc due to its brightness, but we see that it is completely covered by retinal veins and arteries, so we decided to apply the Gaussian filter also to the optic discs and we gained a 5% more precision in recognition.

When applying the algorithm to the macula we lose some precision due to it's form, as we can see also the human eye has difficulties in identifying it cause it hasn't perfectly defined shape.

Other techniques were applied to the image to get better results such as sharpening, increase of contrast with CLAHE (Contrast Limited Adaptive Histogram Equalization) and other MATLAB's image processing functions, was also used the "activecontour" function to apply the region growing technique for the segmentation of the optic disc but all these techniques gave no improvements, in some cases they even worsened the results of the segmentation.

Ultimately, the main trouble is that the algorithm can't adapt to all the test images so another effective approach could be the use of the Deep Learning's foundational sate-of-the-art Convolutional Neural Networks (CNNs), selected, tailored, and trained to identify image's features.

Bibliography

[1] «MATLAB's Image Processing Toolbox Documentation» [Online].
Available: https://it.mathworks.com/products/image.html.

[2] «Whats is YCBCR? » [Online].
Available: https://www.mir.com/DMG/ycbcr.html

[3] «MATLAB's Gaussian Filter» [Online].
Available: https://it.mathworks.com/help/images/ref/fspecial.html?s_tid=doc_ta.

[4] «Morphological Opening» [Online].
Available: https://homepages.inf.ed.ac.uk/rbf/HIPR2/open.htm.