School of Computer Science and Applied Mathematics
UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

APPM4058A & COMS7238A: Digital Image Processing
Course Project

Mohammed Gathoo / COMS Hons
Nico Jared Govindsamy / COMS Hons

19 June 2022

Iris Segmentation

1 Introduction

The aim of this project is to detect the average colour of the iris in an image of an eye. To accomplish this, we need to first segment the iris from the image of the eye, which constitutes the bulk of this project. According to the World Atlas, the 6 most common colors of the human iris are:

- Brown: 70% - 79%
- Blue: 8% - 10%
- Hazel: 5%
- Amber: 5%
- Gray: 3%
- Green: 2%

Our goal is to predict one of these colours given an image of an eye.

2 Methodology

The first part of the project was to find an appropriate data set. That will be elaborated upon in the Data subsection. The second part of the project is to segment the iris from the entire image of the eye. We will delve into the implementation and methodology for that in the Iris Segmentation subsection. Finally, we will discuss the colour prediction technique in the Average Colour Prediction subsection.

2.1 Data:

The data being used in the project were various colour images of the human eye (as well as a few animal eyes). The images are predominantly that of close-up images of an eye, as wells as a handful of images containing larger portions of the face and images containing two eyes within the same picture. The file type used was *.jpg*.
The images were classified into 6 groups of each colour, and the appropriate images were read from their respective groups when testing for a specific colour. Further details about choice of data set are disclosed in the Experimental Setup section of the report.

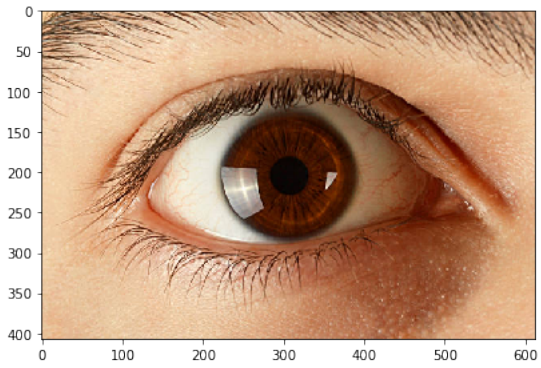

Figure 1: Input Image

2.2 Iris Segmentation:

The scikit-image (or skimage) library provides us with a multitude of functions and methods to help with the implementation of our solution. In order to read in the images for use, the 'io' module from skimage was used to read the images and subsequently convert them to NumPy arrays for manipulation

```
test_img = io.imread("brown/brown6.jpg")
io.imshow(test_img)
```

The colour image was then converted to a grayscale image using the color module from skimage and a Gaussian smoothing filter was applied to the grayscale image in order to remove noise present in the image. We then perform binary thresholding on the image to attempt segmenting the pupil of from other objects in the image. The reason this is done is because we assume the pupil of the eye to be the same colour black and having the darkest intensity in the image.

```
gray_img = color.rgb2gray(test_img)
gray_noiseless = filters.gaussian(gray_img,
    1)
min_intensity = np.min(gray_noiseless)
if min_intensity < 0.1:
    curr_th =  0.05 + 2*min_intensity
else:
    curr_th = 0.05 + min_intensity
img_th = gray_noiseless < curr_th
```

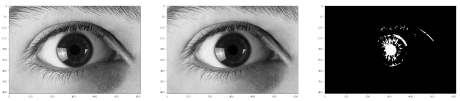

Figure 2: Pre-processing Images

2.2.1 Detecting the pupil:

Because we assume the pupil will always be the darkest area of the eye, we set a threshold intensity to filter the pixels of the grayscale noiseless image such that any intensity greater than that threshold will be highlighted (or set to true) in order to indicate areas of focus.
Due to the assumption we made above, sometimes the binary thresholding will leave other black areas intact that are not the pupils in the image, such as dark eyebrows or eyelashes. We have not completely fixed this, but a fix that works sometimes is drawing a line from the middle of the image downwards to detect if there are eyebrows or eyelashes and then removing all those white values that aren't the pupil pixel values. We first perform binary dilation before drawing this line so that the line will be able to detect the white pixels.
We then continuously apply binary erosion to this thresholded image in order to further remove any pixels that aren't within the pupil and don't match the structuring element of a disk. The purpose of the continuous erosion is to remove objects in the binary image that are not circular like the pupil, leaving only the pupil pixels remaining.

```
def dilation_preprocess(img_th):
    dilated_img = deepcopy(img_th)
    for i in range(2):
        dilated_img = morphology.
            binary_dilation(dilated_img)

    coords_dil = np.where(dilated_img == 1)
    average = np.average(coords_dil,1)

    center = np.array([img_th.shape[0]/2,
        img_th.shape[1]/2])
    center_radius = np.abs(center[1]-
        average[1])
    center_radius_x = np.abs(center[0]-
        average[0])
    if center_radius < 10: center_radius =
        0.2*img_th.shape[1]
    if center_radius_x < 10:
        center_radius_x = 0.2*img_th.shape
        [0]
    turning_point = (center+average)/2

    for x in range(img_th.shape[0]):
        for y in range(img_th.shape[1]):
            if (x < turning_point[0] - 2.5*
                center_radius_x) or (x >
                turning_point[0] + 2.5*
                center_radius_x):
                dilated_img[x][y] = 0
            if (y < turning_point[1] - 2.5*
                center_radius) or (y >
                turning_point[1] + 2.5*
                center_radius):
                dilated_img[x][y] = 0

    return dilated_img


#perform circle hit-or-miss on a
    thresholded image to find pupil
def erosion_preprocess(dilated_img):
    disk = morphology.disk(2)
    eroded = dilated_img
    coords = np.where(eroded == 1)
    while len(coords[0]) > 500:
        eroded = morphology.binary_erosion(
            eroded,disk)
        coords = np.where(eroded == 1)
    #io.imshow(eroded)

    return coords
```

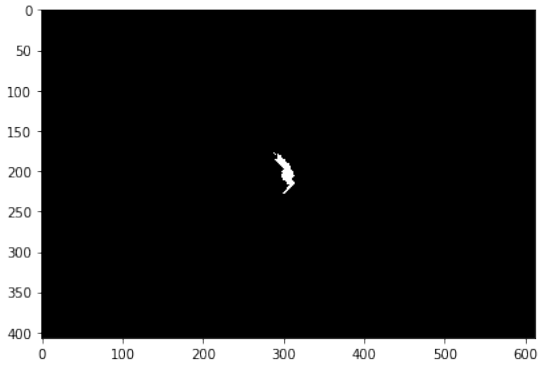

Figure 3: Eroded Image

We then approximate the position of the pupil by first cropping out a portion of the image we believe to contain the pupil and centering it around the pixels we believe fall directly on the pupil. This part of the code also takes into

consideration if two eyes are in the image by perform a simple clustering algorithm in which two groups are formed and one group is considered (a single eye).
The resulting cropped version of the image will the contain the entire eye and not just the pupil.

```
#find approximate position of pupil
group_a = []
group_b = []

#performed distance measurements to
    separate pixels into two groups

radx = int(0.12*test_img.shape[0])
rady = int(0.12*test_img.shape[1])
xavg = int(np.floor(xavg))
yavg = int(np.floor(yavg))

eye_img = test_img[xavg-radx:xavg+radx,yavg
    -rady:yavg+rady]
```

From the skimage.features module, we use 'feature.canny', and edge detection algorithm in order to detect the edges around the iris, and then use a Hough transform, namely the circular Hough transform
(hough_circle and hough_circle_peaks) to find the edges that outline the iris.

```
eyeEdges = feature.canny(eye_threshold)
a = eye_img.shape[0]
b = eye_img.shape[1]
houghRadii = np.arange(int(a/6),int(a/2),
    1)
houghRes = hough_circle(eyeEdges,
    houghRadii)
accums, cx, cy, radii = hough_circle_peaks(
    houghRes, houghRadii, total_num_peaks
    =1)
```

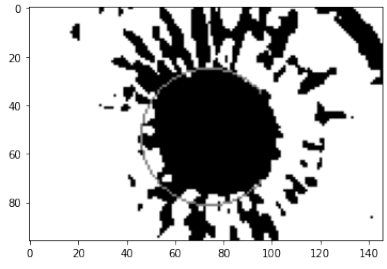

Figure 4: Hough circle image

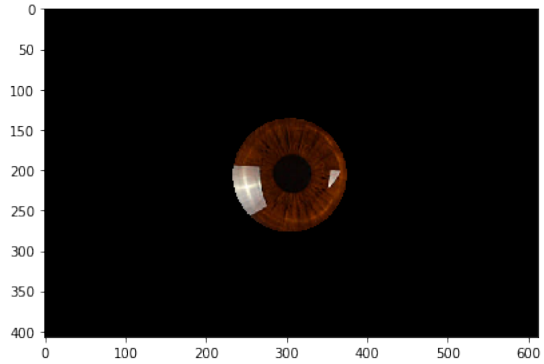

Figure 5: Iris image

2.2.2 Removing Pupil from Eye:

If we are to include the intensities of the pupil in our calculation of the average intensity of the eye iris, it will sway our results, specifically, the output predicted image will be darker than expected. There is also possible glare that needs to be removed to prevent further swaying of the results, which will make the predicted colour brighter. As such, we remove those pixels from the average intensity calculation. This is done by performing binary thresholding again, except this time we remove all pixels that do not fall within a specified range (i.e. very bright or very dark pixels are removed from the image), leaving only coloured pixels in the final image.

```
gray_iris = color.rgb2gray(iris_img)
eye_threshold = deepcopy(gray_iris)
for i in range(gray_eye.shape[0]):
    for j in range(gray_eye.shape[1]):
        if gray_iris[i][j] < 0.12 or
            gray_iris[i][j] > 0.7:
            eye_threshold[i][j] = 0
        else:
            eye_threshold[i][j] = 1
```

2.3 Average Colour Prediction:

From the resulting image with only the coloured pixels highlighted, we average the intensities by summing all and dividing the total by the number of pixels remaining. The resulting color is then displayed.

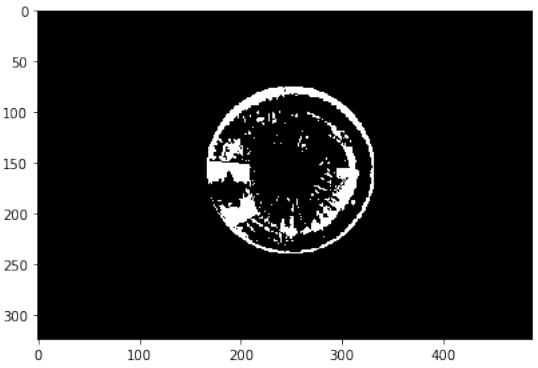

Figure 6: Threshold Eye image

```
avg_colour = [0,0,0]
count = 0
output_img = deepcopy(eye_img)
for i in range(eye_img.shape[0]):
    for j in range(eye_img.shape[1]):
        if (eye_threshold[i][j]):
            avg_colour += eye_img[i][j]
            count += 1
        else:
            output_img[i][j] = [0,0,0]
avg_colour = avg_colour/count

colour = np.zeros((3,3,3))
for i in range(len(avg_colour)):
    avg_colour[i] = avg_colour[i]/255
colour[1][1] = avg_colour
```

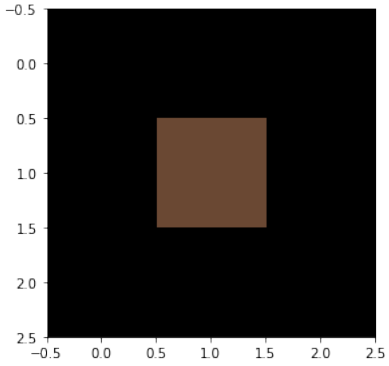

Figure 7: Output image

```
def predict_eye_color(rgbval):
    con1,con2,con3 = 0 < np.abs(rgbval[0]-
        rgbval[1]) < 15,0 < np.abs(rgbval
        [1]-rgbval[2]) < 15,0 < np.abs(
        rgbval[0]-rgbval[2]) < 15
    con4 = rgbval[0] > 150 and rgbval[1] >
        150 and rgbval[2] > 150
    if con1 and con2  and con3 and con4:
        return "gray"
    elif rgbval[2] > rgbval[1] and rgbval
        [2] > rgbval[0]: #blue component
        greater than red and green
        compeonents
        return "blue"
    elif rgbval[1] > rgbval[0] and rgbval
        [1] > rgbval[2]: #if green
        component is greater than the other
         two
        return "green"
    else: #red component is greatest
        if rgbval[2] < 10 and rgbval[0] >
            150:
            return "amber"
        elif rgbval[0] < 100 or rgbval[2] <
             60 and rgbval[1] < 80:
            return "brown"
        else:
            return "hazel"
```

3 Experimental Setup

When choosing our data set, we considered a number of different scenarios. Since we struggled to find a colour eye image data set from the internet, we made our own using images from Google images. We selected around 15-20 images for each colour and labelled the images according to their colour as well as sorted them into folders. This way, we not have the true labels. The images were selected to handle these cases (check appendix):

- Images of a single close-up eye relatively in the middle of the image.
- Same as above but not in the middle.
- Images where there are two eyes.
- Images where there is makeup in the image.
- Images where the eye is not facing the camera directly.

We obtained a few images of the same scenario to make decent comparisons in the results obtained from applying the algorithm to different eye colours.

4 Evaluation

Due to the assumptions we made in designing and implementing our algorithms, there are several cases where the iris will not be segmented properly and so results in an inaccurate prediction of colour because the intensities are calculated for a wrong area of the image (instead of the iris). These cases are when there are other dark areas in the image, especially when they are larger in number of pixels than the pupil itself, and also when the pupil isn't dark itself (this can arise from glare increasing the intensity of the pupil which results in a pupil that isn't dark).

These can occur when shadows, dark makeup, dark eyebrow or dark eyelashes exist in the image. Since we used binary thresholding, morphological operations and averaging calculations to estimate the position of the pupil under the assumption that they were a certain intensity and darker than everything else, this means that those other things can be falsely detected as the pupil and so the algorithm doesn't work. Another case in which our code does not work is when the pupil is larger than the Hough circles can detect. This is because of the way we approximate the pupil and iris sizes from the image size.
Our algorithm performed poorly in predicting gray, amber and green iris. Gray eyes typically consist of other colours in the iris like green, blue or hazel which makes the prediction more difficult as the average colour intensity is askewed by the other colours evident. Therefore, we found it harder to predict gray eyes.

The results are all in the appendix. The images on the left are the originals and the images next to them are the segmented iris images. The statistics for correct segmentation of the iris are as follows:

- amber: 3/11 - 27%
- blue: 9/10 - 90%
- brown: 10/20 - 50%
- gray: 14/18 - 78%
- green: 7/18 - 39%
- hazel: 9/18 - 50%

```
amber   total number images:  11      total predicted correct:  1    success rate:  9.090909090909092
blue    total number images:  11      total predicted correct:  11   success rate:  100.0
brown   total number images:  20      total predicted correct:  17   success rate:  85.0
gray    total number images:  18      total predicted correct:  1    success rate:  5.5555555555555555
green   total number images:  18      total predicted correct:  5    success rate:  27.777777777777778
hazel   total number images:  18      total predicted correct:  13   success rate:  72.22222222222221
total success rate:  0.5
```

Figure 8: Results

5 Appendix

Figure 9: output for amber colour eyes

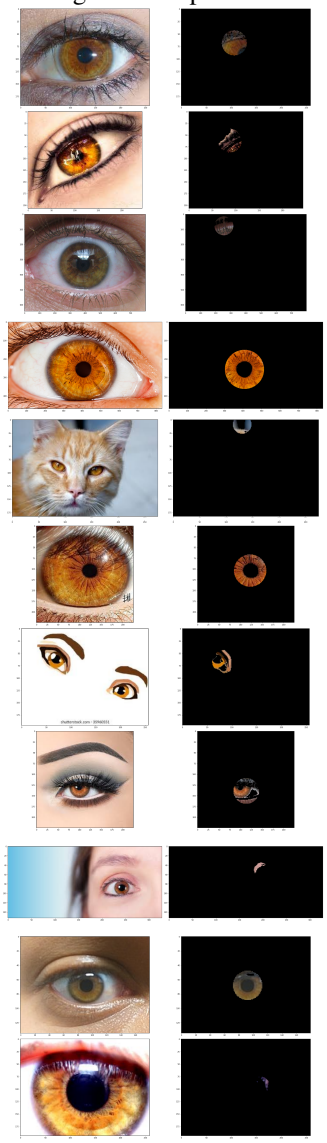

Figure 10: output for blue colour eyes

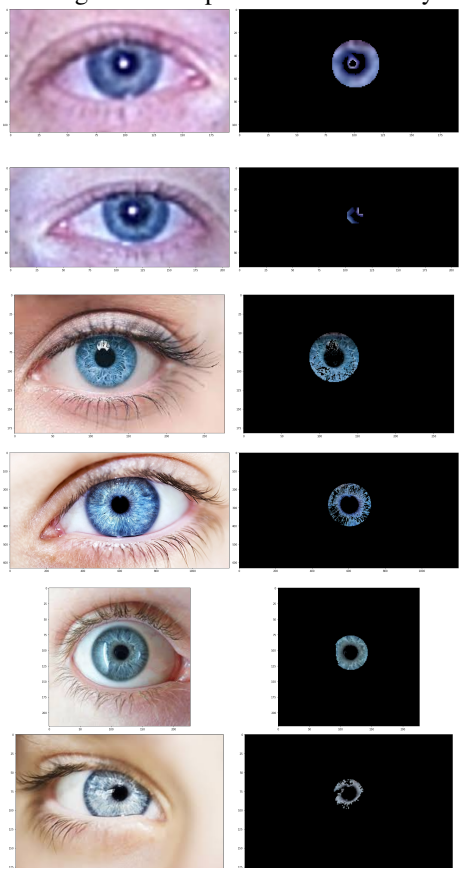

Figure 11: output for blue colour eyes

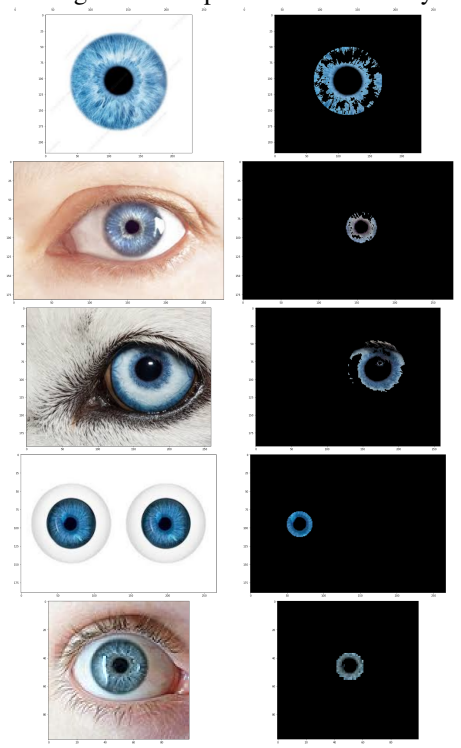

Figure 12: output for brown colour eyes

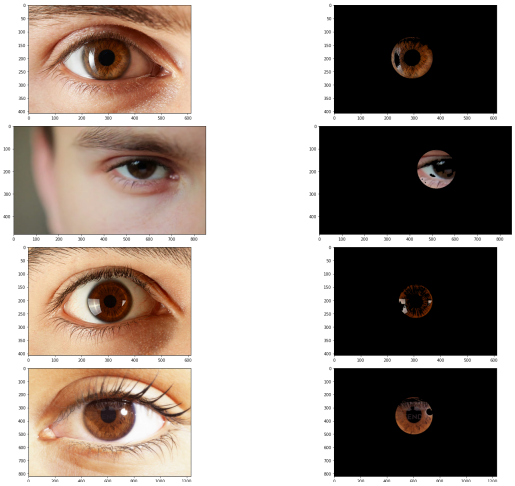

Figure 13: output for brown colour eyes

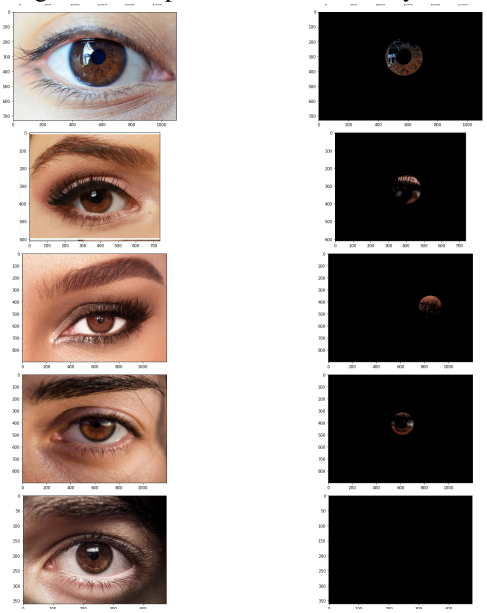

Figure 14: output for brown colour eyes

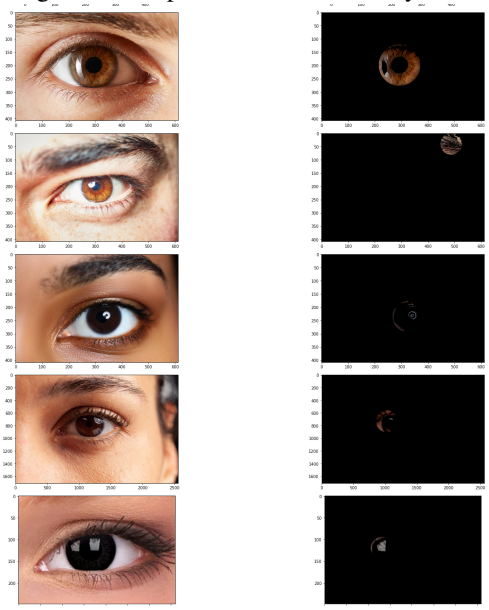

Figure 15: output for gray colour eyes

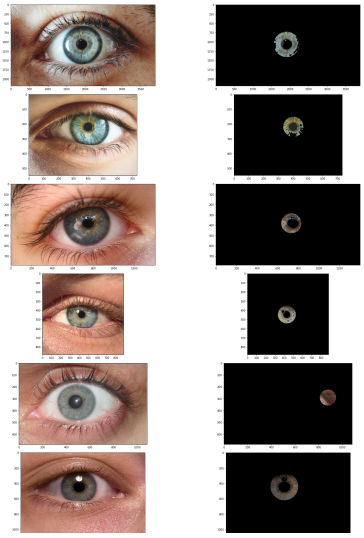

Figure 16: output for gray colour eyes

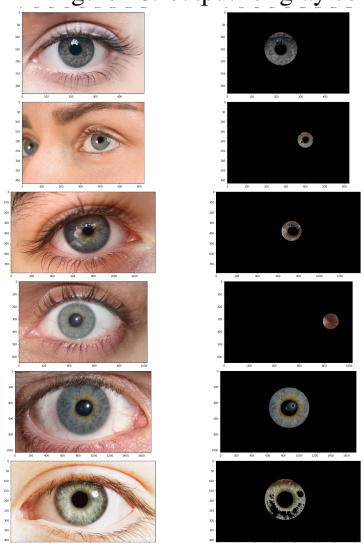

Figure 17: output for gray colour eyes

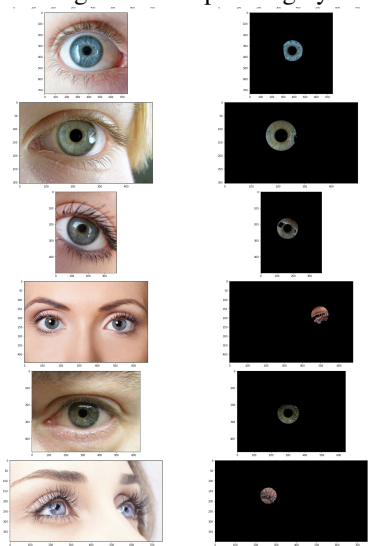

Figure 18: output for green colour eyes

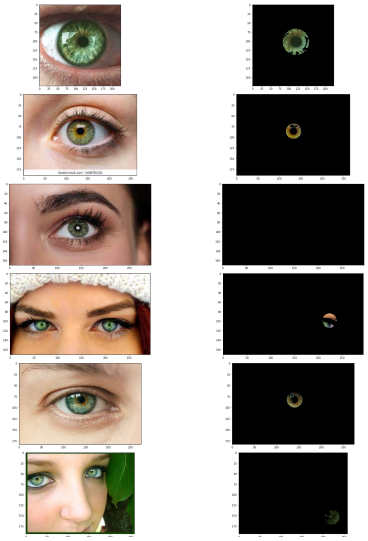

Figure 19: output for green colour eyes

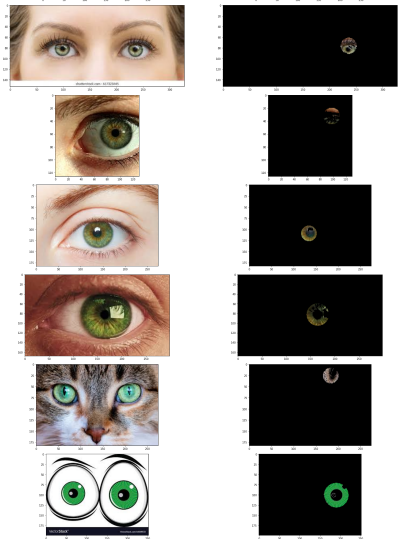

Figure 20: output for green colour eyes

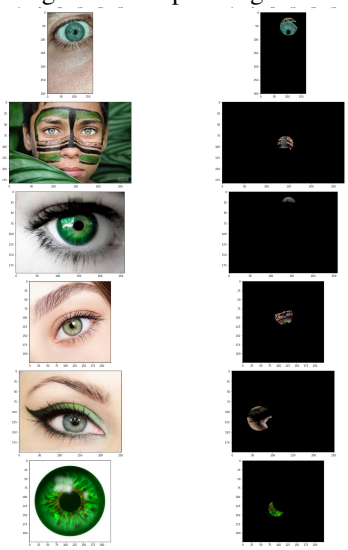

Figure 21: output for hazel colour eyes

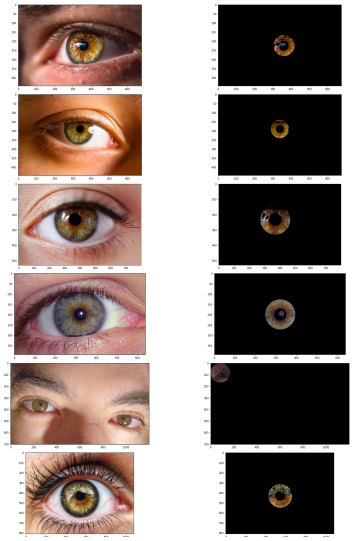

Figure 22: Highlighted pixels not with very high or low intensities (taken from grayscale image), output for hazel colour eyes

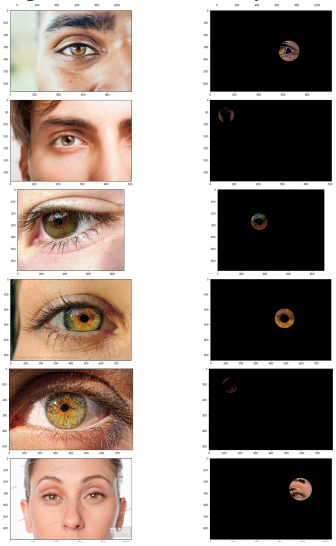

Figure 23: output for hazel colour eyes