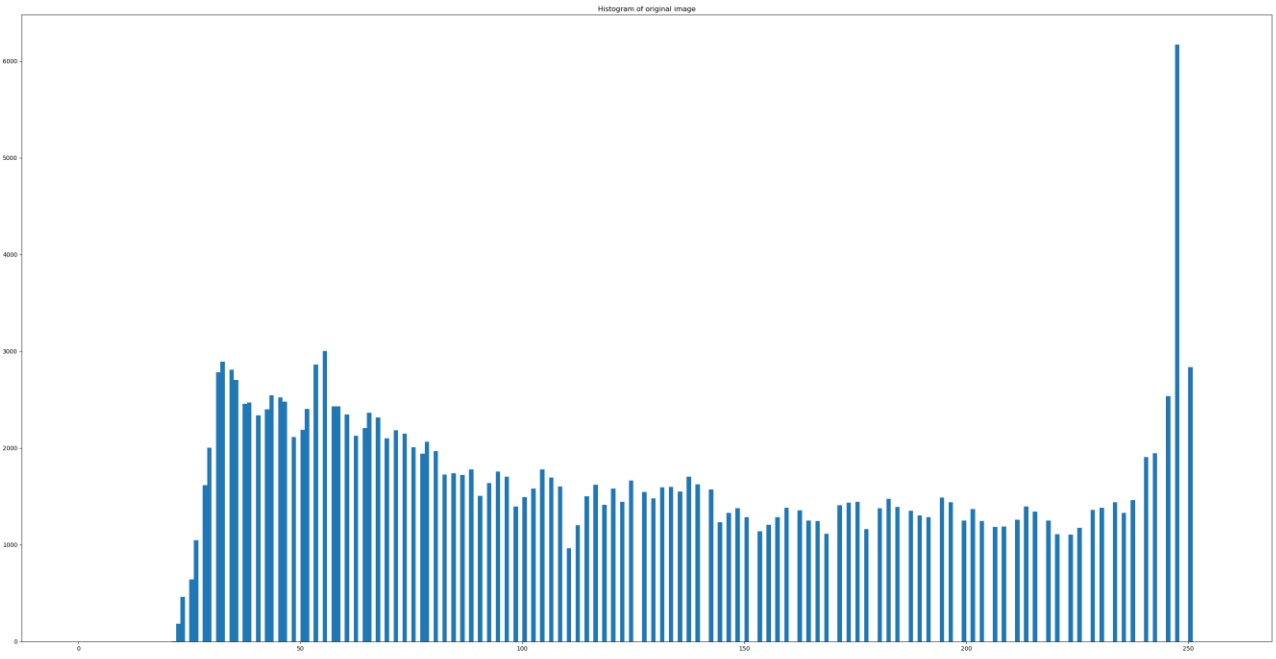
COMPUTER VISION (CSCI -631)

Homework-1 **Name: Rajkumar Lenin Pillai**

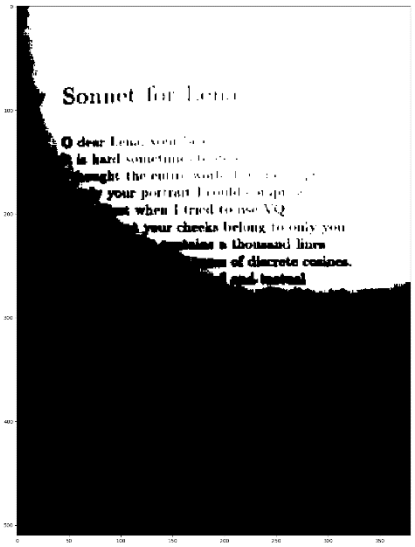
Question-1

Solution:

a.) Histogram of original Image:-



b.) The results after setting the value to 255 at a global threshold value=130

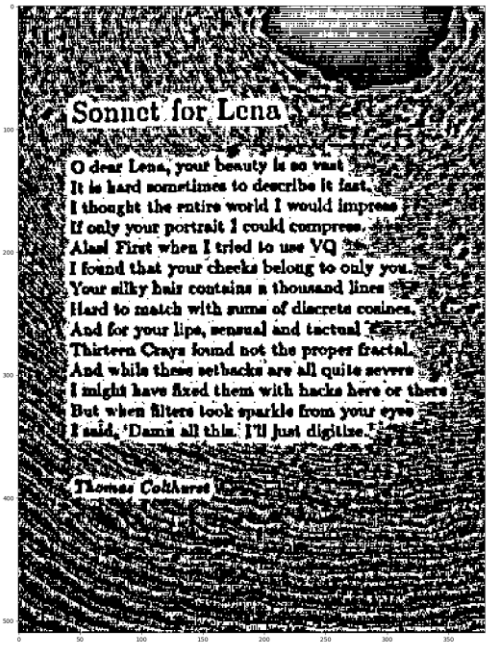


The above image is the output when the global threshold value is considered to be 130 which is obtained by observing the histogram of the original image. The Binary thresholding method was used and traversing through every pixel of image, if the value is greater then 130 then it is set to 255 else it is set to 0.

If the image is a bimodal image then the global threshold value can be obtained by using the below code which is included in the q1.py file in the comments. The below code when used gives the global_threshold_value of 131 which is approximately equal to the global_threshold_value that is considered. In the below code at the first line the default threshold value used is 0 (second argument of cv2.threshold function) but the function uses OTSU's method of Binarization and computes the global_threshold_value.

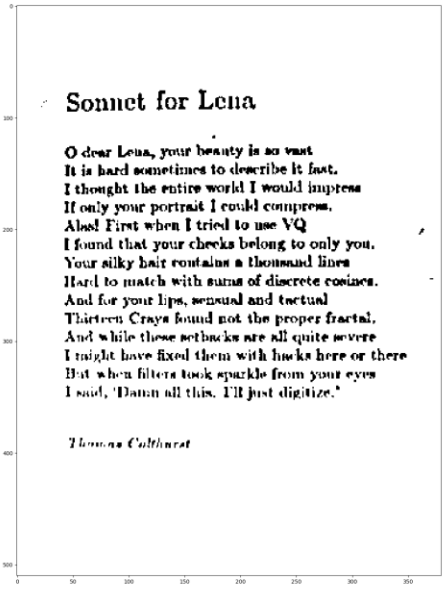
```
global_threshold_value,thresholded_image = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
print('Global Threshold Value',global_threshold_value)
plt.imshow(thresholded_image, 'gray')
plt.show()
```

c.) Results after using adaptive thresholding using window size 11 x 11



The adaptive thresholding formula which was used is t = mean(N ×N) + C .Here initially the value of C was taken as 0. It succeeds in separating the foreground and background and makes the text visible because the value of mean which is computed using the formula t = mean(N ×N) is between the value of intensities of background and foreground pixels. On the other areas which is the margin, the mean value doesn't seem to be a good threshold value because the intensity values of neighboring pixels are almost similar and mean is almost equal to the center pixel. To handle this problem the value of C was considered to be 7 in the above mentioned formula, so that all pixels in the image which are along the margin can be shown as background. The results after using mean adaptive thresholding with window size 11 x 11 and C= 7 is shown below:

d.) Results using mean adaptive thresholding using 11 x 11 window and C =7



The above image shows good separation between background and foreground in the entire image with adaptive thresholding using mean and is not effected by illumination in the image.

Question-2

Solution:

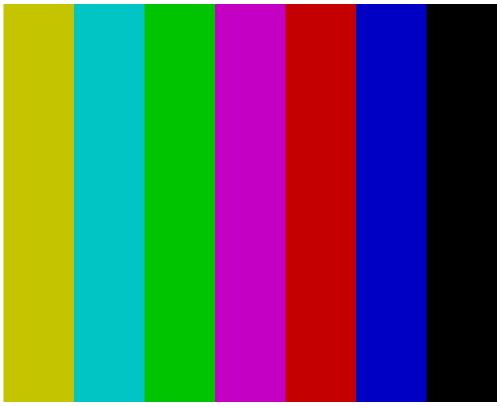
a.) The Alice image with blue channels set to zero.

The code to generate above output is present in file q2a.py

The above image is generated in the code by making a copy of image and setting it's blue channel to zero and is stored in file Alice-no-blue.png. The third value in the image array corresponds to BGR channel so to set blue channel to zero the first element which is at 0^{th} position is set to zero in the python code. The output image is stored in file"Alice-no-blue.png"

b.) RGB to Grayscale:

i.) Original swatch



ii.) RGB to Grayscale using an equally weighted mean

iii.) RGB to Grayscale using relative luminance

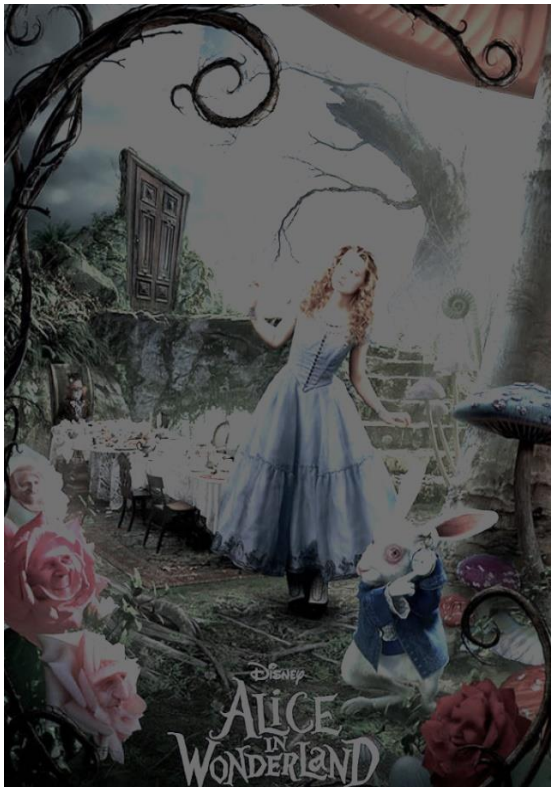
The equally weighted mean and the relative luminance method yield different result. Comparing the output image of both these methods the output of relative luminance seems to be correct.

The equally weighted mean is a simple technique in which we take average of three colors red , green and blue but the problem arises due to the fact that we don't consider the contribution of these colors. According to formula we are computing grayscale pixel by formula 0.33*R+0.33*G+0.33*B . Here we are assuming that all colors contribute in equal proportion but that's not real. So the average method to convert RGB to grayscale fails in some cases. On the other hand , relative luminance takes into account the contribution of color. The formula used in relative luminance is 0.299 * R + 0.587 * G + 0.114 * B.

c.) The result of shitingIm() function on original Image:

The code to generate above output is present in file q2c.py . For the above output the value k was taken as 0.45 and input channel was 1 which is green so the value was added to all green channel pixels .The output image is stored in file " Alice-Shift-Image.png".

d.) The output image after clamping in range 0 to 120



The code to generate above output is present in file q2d.py . Pixels of an image which are outside certain range can be forced to fall within that range. In this example the values of pixels are restricted within the range of 0 to 120. The output of above image which is obtained after performing the clamp operation on the input image , we can observe that the bright portion of original image appear more bright and certain details in the image are also missing since their pixel values are exceeding the range of 0 to 120. The output image is stored in file ”Alice-clamp.png”.

e.) Alice image RGB to HSV:

The above image is obtained after converting the BGR 'Alice image' to HSV using opencv built-in package.

Modified HSV image after increasing the saturation (HSV back to RGB) :

The code to generate above output is present in file q2e.py . Saturation in the image was increased by value 0.20 and the output image is stored in file ” Alice-rgb-hsv.png” . Saturation shows the purity of color in the image. The value 0.2 is within the range mentioned in question and it makes the colors in the image more dark.

Increasing the saturation by value of 0.6 gives the below result:

This shows that colors become darker as the saturation is increased and if saturation is decreased then it drains the purity of color in the image. We can change the skin tone of a person in image by adjusting the saturation accordingly.