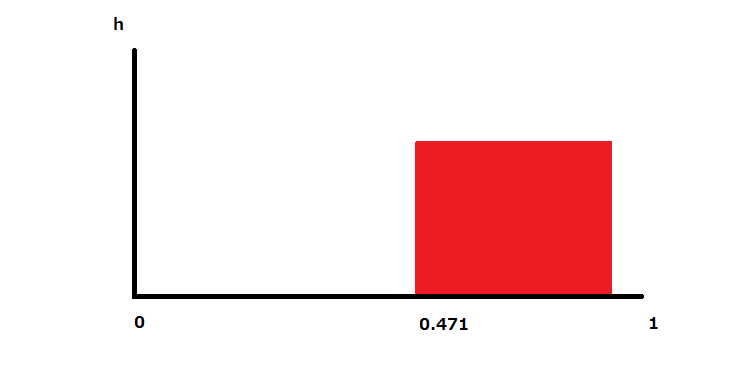
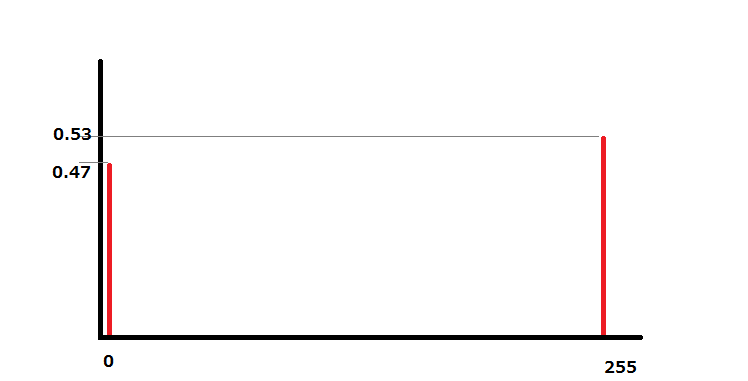
**NUMBER 1**

Using a Look up Table is almost always much more efficient than calculating the map using single pixels. Because in the approach of pixel by pixel the program has to calculate n^2 square root calculations along with n^2 assignments, while the LUT on needs to do 256 square root calculations and n^2 assignments making it significantly faster. However it is recommended to use LUT when grey levels are L<n^2

**NUMBER 2**



**NUMBER 3**

From 0 to 63 and 128 to 191 all pixels will be converted to black  
From 63to 127 and from 192 to 255 will be converted to white  
The histogram looks something like this 

**NUMBER 4**

The method I decided to use was keypoint comparison

1. a = imread('IMAGENAME1.jpg');
2. b = imread('IMAGENAME2.jpg');
3. %load array of pixels a
4. %load array of pixels b
6. sizeA = size(a);
7. sizeB = size(b);
9. %find the height and width of both images
10. hA = size(a,1);
11. wA = size(a,2);
12. hB = size(b,1);
13. wB = size(b,2);
15. %find at what distance should a new point be looked at
16. scaleHA = floor(hA/10);
17. scaleWA = floor(wA/10);
18. scaleHB = floor(hB/10);
19. scaleWB = floor(wB/10);
21. %declaring variables for the loops
22. similarity = 0;
23. rowCount = 0;
24. colCount = 0;
26. rowLocationA = 0;
27. rowLocationB = 0;
28. colLocationA = 0;
29. colLocationB = 0;
31. %While loop checks if all ten rows have been looked at, if not it goes down
32. %the level to a nested while loop
33. %nested while loop checks if 10 columns of the row have been checked if not
34. %it compares two seleceted points on each image, if it is a match then adds
35. %1 to similarity score, adds that row to checked and moves on
36. %loops until all ten rows and columns were checked
37. while rowCount<10
38. rowLocationA = rowLocationA + scaleHA;
39. rowLocationB = rowLocationB + scaleHB;
40. %disp(rowLocationA)
41. %disp(rowLocationB)
42. while colCount<10
43. colLocationA = colLocationA + scaleWA;
44. colLocationB = colLocationB + scaleWB;
45. if( a(rowLocationA,colLocationA) == b(rowLocationB,colLocationB))
46. similarity = similarity + 1;
47. end
48. colCount = colCount + 1;
49. end
50. colLocationA = 0;
51. colLocationB = 0;
52. colCount = colCount - 10;
53. rowCount = rowCount+1;
54. end
56. disp(similarity)

For my method I decided to select 100 points on each image and compare them between each other. On paper I saw this as a relatively good solution because I first believed that comparing color is enough to determine similarity, unfortunately it does not work so well for images which are exactly the same except for the color.

Here are the similarity scores I got using the above code and the images provided on A2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **A** | **B** | **C** | **D** |
| **A** | **100** | **10** | **78** | **1** |
| **B** | **10** | **100** | **12** | **0** |
| **C** | **78** | **12** | **100** | **1** |
| **D** | **1** | **0** | **1** | **100** |

One solution which could possibly solve the situation that happens when comparing B and A, where the image is exactly the same but with slightly different shading is instead of using the exact color it can compare color within a small radius on the color scale.

5) Using nearest neighbor extrapolation enlarge image

a = imread('A.jpg');

k = SIZE;

ogSize = size(a); % find the size of the inputed image

sizeMod = [k k]; % size modifier kW and kH

newSize = max((sizeMod.\*ogSize(1:2)),1); %find the new size based off scale

%Using nearest neighbor extrapolation compute the pixels to expand

rowEx = min(floor(((1:newSize(1))-1)./sizeMod(1)+1),ogSize(1));

colEx = min(floor(((1:newSize(2))-1)./sizeMod(2)+1),ogSize(2));

%plug in extrapolated values into old image

newImage = a(rowEx,colEx,:);

imshow(newImage)

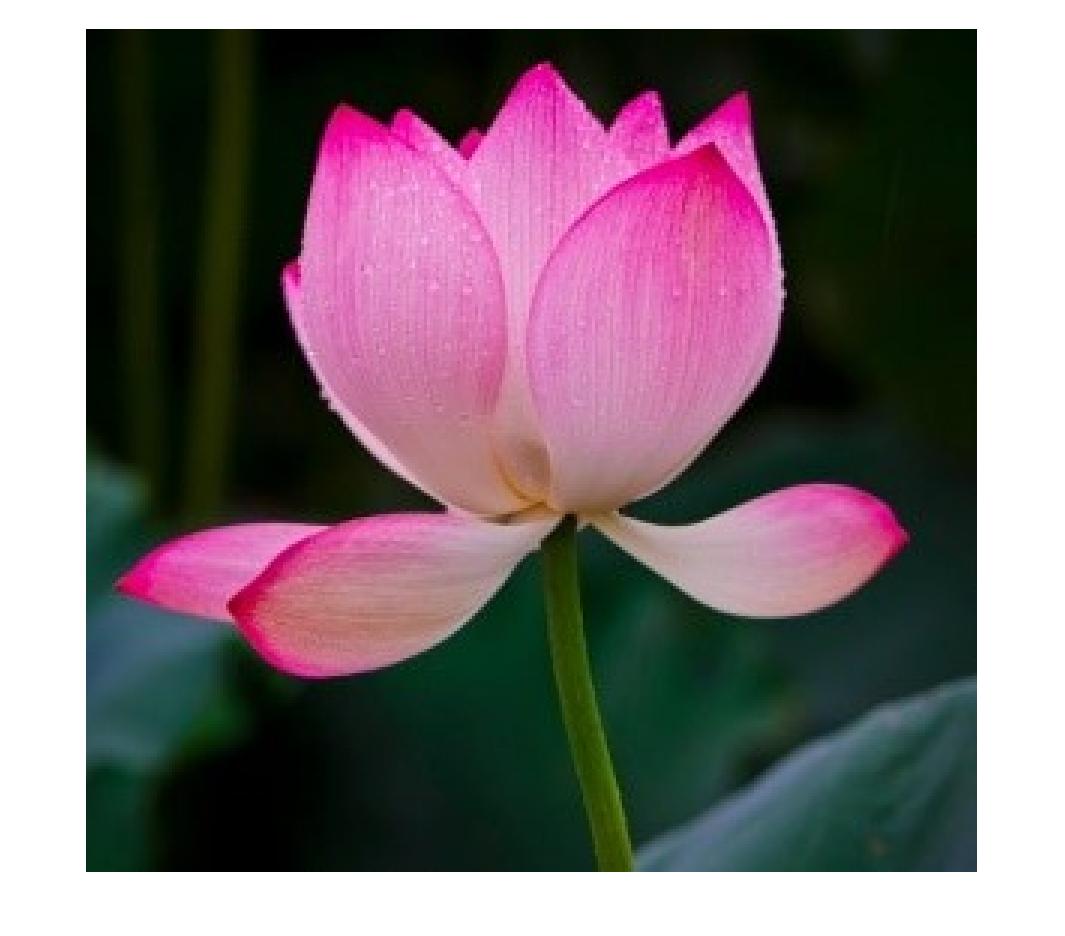


This is the original image used, its resolution is 471x374 and its file size is 13.8KB

After the above code was ran with the “SIZE” replaced by a 2 it produced the following image.



This image’s resolution was increased to 768x655 and its size increased to 34.8KB.

Lastly once the code was ran with k being equal to 3 the following image was produced.

This images resolution became 1065x936 and its size became 63.1KB.