

CS-663 Assignment 1 Q2

Soham Naha (193079003)
Akshay Bajpai (193079002)
Mohit Agarwala (19307R004)

September 1, 2020

2 (55 points)

Input images:

1. 2/data/barbara.png
2. 2/data/TEM.png
3. 2/data/canyon.png
4. 2/data/retina.png
5. 2/data/church.png
6. 2/data/chestXray.png
7. 2/data/statue.png

Image (4) has an associated binary image *2/data/retinaMask.png* indicating the foreground region. All the processing on image (4) should be performed only using the intensities in the foreground region. Image (4) also has an associated reference image *2/data/retinaRef.png* and its associated binary image *2/data/retinaRefMask.png* which are required for part (d) of the question only. Assume the pixel dimensions to be equal along both axes, i.e., assume an aspect ratio of 1:1 for the axes. For the color images, apply the analysis independently to each channel (Note: this is a sub-optimal way of processing color images, in general; some of the reasons for which will be evident from the results that you will get).

2.1 (2 points) Foreground Mask

Find a binary mask for the foreground region for image (7).

- Write a function *myForegroundMask.m* to implement this.
- Display the original image, the binary mask and the masked image.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import matplotlib.image as mpimg
4 import matplotlib as mpl
5 import cv2
6
7
8 def myForegroundMask(input_file, cmap="gray", offset=2):
9     name = input_file.split(".")[2]
10    input_image = cv2.imread(input_file, 0)
11    new_image = np.zeros_like(input_image)
12
13    # PLOTTING PARAMETERS
14    parameters = {'axes.titlesize': 10}
15    plt.rcParams.update(parameters)
16
17    print(np.max(input_image))
18
19    r, c = input_image.shape
20
```

```

21 # mask formation
22 for i in range(input_image.shape[0]):
23     for j in range(input_image.shape[1]):
24         if input_image[i][j] < np.mean(input_image)+offset:
25             new_image[i][j]=0
26         else:
27             new_image[i][j]=1
28
29 # masked image formation
30 masked_image = new_image*input_image
31
32
33 fig,axes = plt.subplots(1,3, constrained_layout=True)
34
35 axes[0].imshow(input_image,cmap="gray")
36 axes[0].axis("on")
37 axes[0].set_title("Original Image")
38
39 axes[1].imshow(new_image,cmap="gray")
40 axes[1].axis("on")
41 axes[1].set_title("Foreground Mask(th = 2)")
42
43 im = axes[2].imshow(masked_image*2,cmap="gray")
44 axes[2].axis("on")
45 axes[2].set_title("Masked Image")
46
47 fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.45)
48 plt.savefig("../"+name+"ForegroundMask.png",cmap=cmap,bbox_inches="tight",pad=-1)
49
50 cv2.imwrite("../" + name+"Mask.png",new_image)
51 cv2.imwrite("../" + name+"ForegroundMasked.png",masked_image)

```

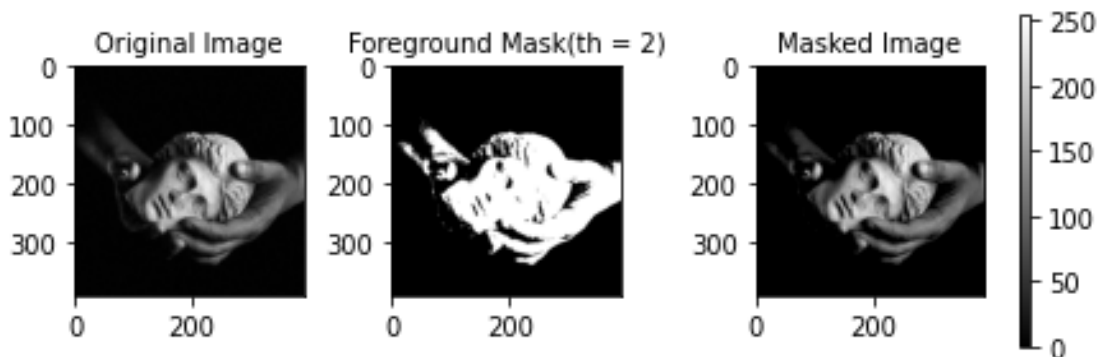


Figure 1: Output of myForegroundMask.py

2.2 (3 points) Linear Contrast Stretching

Design a linear grayscale transformation function to enhance the intensity contrast such that the resulting intensity range is $[0,255]$.

- Write a function *myLinearContrastStretching.m* to implement this.
- Show the formula (or the pseudo code) for the linear function in the report.
- Display the original image and the contrast-enhanced image, without changing the aspect ratio of objects present in the image. Display the colorbar.
- Show the above results on input images 1, 2, 3, 5, 6 and foreground region of image 7 (using the mask generated in part (a)).
- Explain your observations after applying contrast stretching on image (5). Why do you think contrast stretching is or isn't effective here?

Formula used for Linear Contrast Stretching:

$$x = \begin{cases} (s1/r1) \times x, & \text{if } x \leq r1 \\ \frac{(s2-s1)}{(r2-r1)} \times (x - r1) + s1, & \text{if } r1 < x \leq r2 \\ \frac{(255-s2)}{(255-r1)} \times (x - r2) + s2, & \text{if } x > r2 \end{cases} \quad (1)$$

where x is the input pixel intensity, $r1, r2$ are pixel intensities in the input image and $s1, s2$ are pixel intensities in the output image.

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3  import matplotlib.image as mpimg
4  import matplotlib as mpl
5
6  from tqdm import tqdm
7  import cv2
8  from seaborn import distplot
9
10 def plot_hist(input_file, input_image, output_image):
11     """
12     input : input_file_path, input_image, output_image
13     output : saves the histograms for both the images for comparison
14     dependencies : seaborn, numpy, matplotlib
15     """
16     name = input_file.split(".")[2]
17     plt.figure()
18     plt.title("Normalized Histogram Plots for Images")
19     ax = distplot(input_image, color='r', label="Input Histogram",
20                  hist_kws={"alpha": 0.3, "linewidth": 1.5}, bins=256, hist=False)
21     ax = distplot(output_image, color="b", label="Contrast Stretched Histogram",
22                  hist_kws={"alpha": 0.3, "linewidth": 1.5}, bins=256, hist=False)
23     l1 = ax.lines[0]
24     x1 = l1.get_xydata()[ :, 0]
25     y1 = l1.get_xydata()[ :, 1]
26     ax.fill_between(x1, y1, color="red", alpha=0.3)
27     l2 = ax.lines[1]
28     x2 = l2.get_xydata()[ :, 0]
29     y2 = l2.get_xydata()[ :, 1]
30     ax.fill_between(x2, y2, color="blue", alpha=0.3)
31     plt.legend()
32     plt.savefig("../"+name+"LCSHistogram.png", bbox_inches="tight", pad=-1)
33
34
35 def truncate(array):
36     """
37     input : array
38     output : truncated array to make it stay from 0 to 255
39     """
40     r, c = array.shape
41     for i in range(r):
42         for j in range(c):
43             if array[i][j] < 0.0:
44                 array[i][j] = 0
45             elif array[i][j] > 255.0:
46                 array[i][j] = 255.0
47     return array
48
49
50 def myLinearContrastStretching(input_file, x1=[0, 255], x2=[0, 255], cmap="gray"):
51     """
52     input : <input_file_path>, input_image_range(x1), output_image_range(x2), cmap(optional)
53     output : Saves the linear contrast stretched image
54     x1 : [r1, r2] (by default = [0, 255])
55     x2 : [s1, s2] (by default = [0, 255])
56     """

```

```

57 parameters = {'axes.titlesize': 10}
58 plt.rcParams.update(parameters)
59
60 r1,r2 = x1
61 s1,s2 = x2
62 name = input_file.split(".")[2]
63 input_image = cv2.imread(input_file)
64 d = 1
65 if len(input_image.shape)>2:
66     r,c,d = input_image.shape
67 else:
68     r,c = input_image.shape
69
70 if d==1:
71     new_image=np.zeros_like(input_image)
72     for i in tqdm(range(r)):
73         for j in range(c):
74             input_pixel = input_image[i,j]
75             if input_pixel<=r1:
76                 input_pixel = (s1/r1) * input_pixel
77             elif input_pixel>r1 and input_pixel <= r2:
78                 input_pixel = ((s2-s1)/(r2-r1))*(input_pixel-r1) + s1
79             else:
80                 input_pixel = ((255.0-s2)/(255.0-r2))*(input_pixel-r2) + s2
81             new_image[i][j]= input_pixel
82     new_image = truncate(new_image)
83     plot_hist(input_file,input_image,new_image)
84
85 else:
86     input_image = cv2.cvtColor(input_image,cv2.COLOR_BGR2RGB)
87     hsv_image = cv2.cvtColor(input_image,cv2.COLOR_RGB2HSV)
88     h,s,v = cv2.split(hsv_image)
89     v_new = v.copy()
90     for i in tqdm(range(r)):
91         for j in range(c):
92             input_pixel = v[i,j]
93             if input_pixel<=r1:
94                 input_pixel = (s1/r1) * input_pixel
95             elif input_pixel>r1 and input_pixel <= r2:
96                 input_pixel = ((s2-s1)/(r2-r1))*(input_pixel-r1) + s1
97             else:
98                 input_pixel = ((255.0-s2)/(255.0-r2))*(input_pixel-r2) + s2
99             v_new[i,j]= input_pixel
100
101     hsv_image[:, :,2] = v_new
102     plot_hist(input_file,v,v_new)
103     new_image = cv2.cvtColor(hsv_image,cv2.COLOR_HSV2RGB)
104
105
106 fig,axes = plt.subplots(1,2, constrained_layout=True)
107 axes[0].imshow(input_image,cmap="gray")
108 axes[0].axis("on")
109 axes[0].set_title(r"Original Image")
110 im = axes[1].imshow(new_image,cmap="gray")
111 axes[1].axis("on")
112 axes[1].set_title(r"Linear Contrast Stretched Image")
113
114 cbar = fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.45)
115 plt.savefig(".." + name + "LCS.png",bbox_inches="tight",pad=-1)
116
117 plt.imsave(".." + name + "LinearContrastStretching.png",new_image,cmap=cmap)

```

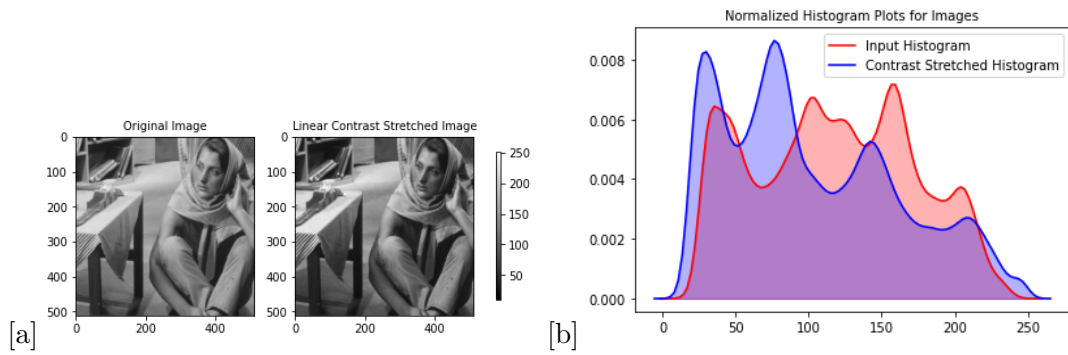


Figure 2: (a) Linear Contrast Stretching for barbara.png (b) Histogram comparison for barbara.png

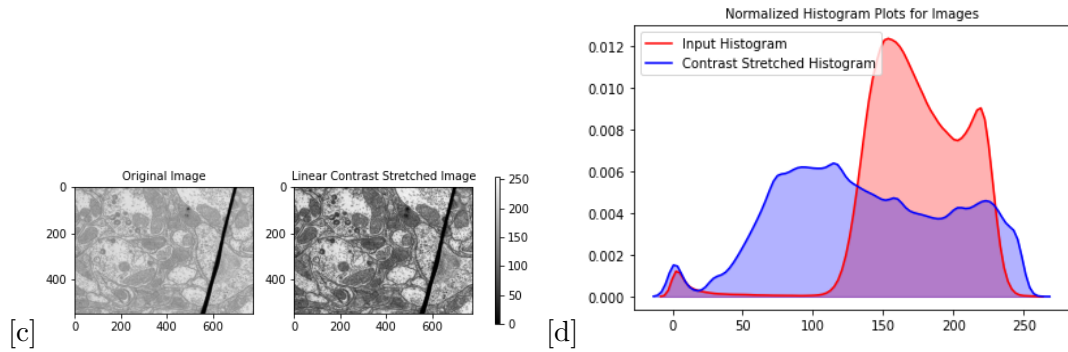


Figure 3: (c) Linear Contrast Stretching for TEM.png (d) Histogram comparison for TEM.png

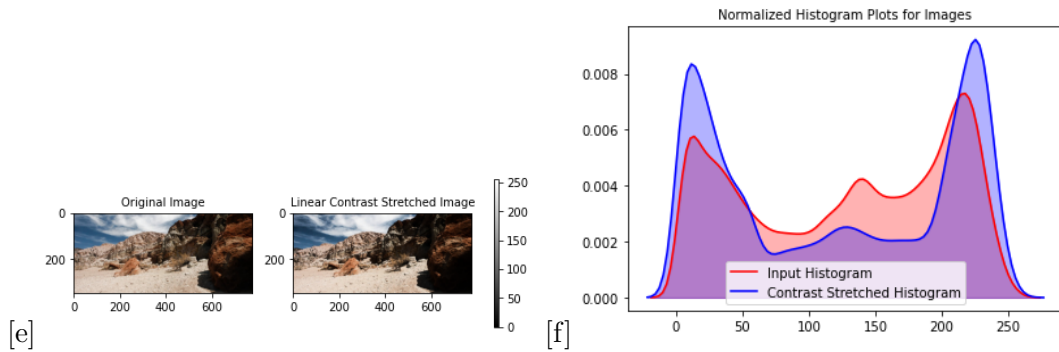


Figure 4: (e) Linear Contrast Stretching for canyon.png (f) Histogram before and after LCS for canyon.png

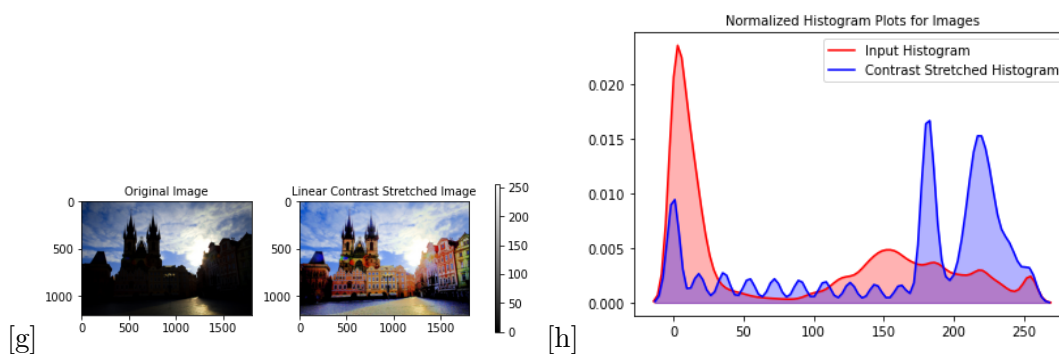


Figure 5: (g) Linear Contrast Stretching for church.png (h) Histogram comparison for church.png

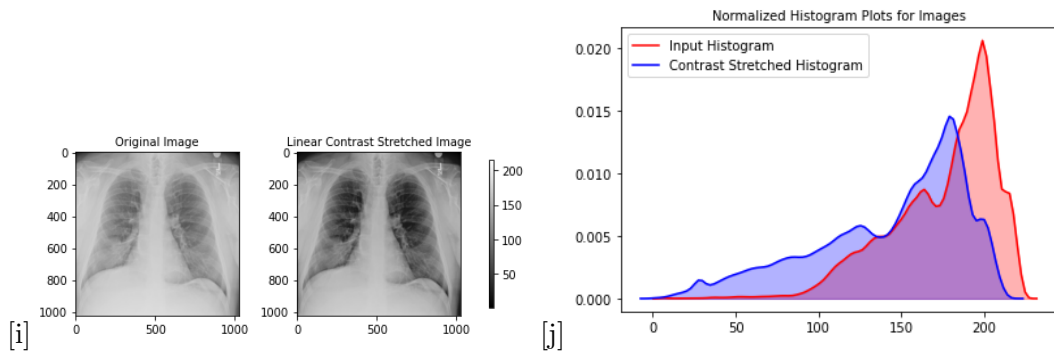


Figure 6: (i) Linear Contrast Stretching for chestXray.png (j) Histogram comparison for chestXray.png

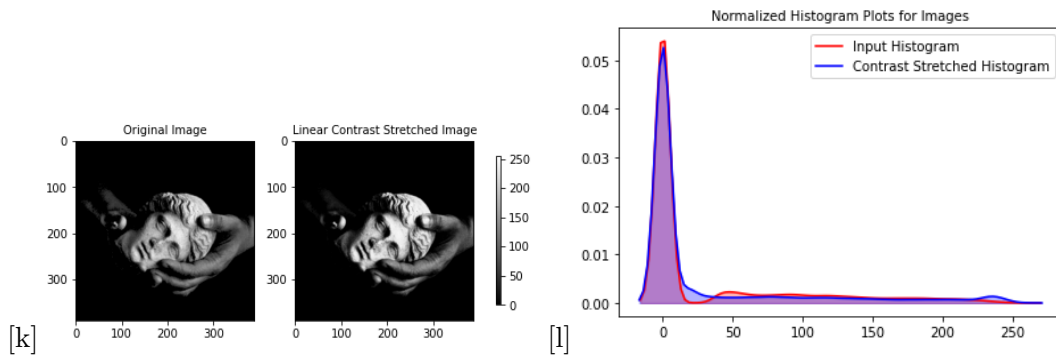


Figure 7: (k) Linear Contrast Stretching for statueForegroundMasked.png (l) Histogram comparison for statueForegroundMasked.png

Observation on applying LCS on *church.png* :

In general, if there are outliers present in the image, it reduces the effect of contrast stretching. In this case, the bright region of the sun, acts as an outlier as most of the image is dark. So, in effect although the colors are brightened but the contrast is not enhanced, as evident from the output histogram at Figure(5)[h].

2.3 (5 points) Histogram Equalization (HE)

Perform (global) HE on the entire image.

- Write a function *myHE.m* to implement this.
- Display the original image and the contrast-enhanced image.
- Show the above results on input images 1, 2, 3, 5, 6 and foreground region of image 7(using the mask generated in part (a))
- Explain your observations after applying HE on image (5). Which one would you prefer to improve image (5)- HE or contrast stretching?

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3  import matplotlib.image as mpimg
4  from numpy import zeros_like
5  from tqdm import tqdm
6  from math import floor
7  import cv2
8  from seaborn import distplot
9
10 def plot_hist(input_file,input_image,output_image):
11     """
12     input : input_file_path, input_image, output_image
13     output : saves the histograms for both the images for comparison
14     dependencies : seaborn, numpy, matplotlib
15     """
16     name = input_file.split(".")[2]
17     plt.figure()

```

```

18 plt.title("Normalized Histogram Plots for Images")
19 ax = distplot(input_image,color='r',label ="Input Histogram",hist_kws={"alpha": 0.3,
20     "linewidth": 1.5},bins=256,hist=False)
21 ax = distplot(output_image,color="b",label ="Histogram Equalized Histogram",
22     hist_kws={"alpha": 0.3,"linewidth": 1.5},bins=256,hist=False)
23 l1 = ax.lines[0]
24 x1 = l1.get_xydata()[:,0]
25 y1 = l1.get_xydata()[:,1]
26 ax.fill_between(x1,y1, color="red", alpha=0.3)
27 l2 = ax.lines[1]
28 x2 = l2.get_xydata()[:,0]
29 y2 = l2.get_xydata()[:,1]
30 ax.fill_between(x2,y2, color="blue", alpha=0.3)
31 plt.legend()
32 plt.savefig("../"+name+"HEHistogram.png",bbox_inches="tight",pad=-1)
33
34
35 def truncateHE(array):
36     """
37     This function truncates the array values to check whether the values
38     are within range of [0,255]
39     """
40     if array<0:
41         array = 0
42     elif array>255.0:
43         array = 255.0
44     return array
45
46 def calculate_CDF(array,maximum,r,c):
47     """
48     This function is used to calculate the CDF of the 2D image.
49     array : For gray scale the whole image is the input and for RGB each
50     of the color slices are the inputs.
51     maximum : maximum pixel intensity of the 2D image
52     output : the CDF of the 2D image
53     """
54     freqs = np.zeros((maximum+1,1))
55     probf = np.zeros((maximum+1,1))
56     cum = np.zeros((maximum+1,1))
57
58     for i in range(r):
59         for j in range(c):
60             freqs[int(array[i][j])]+=1
61
62     for i,j in enumerate(freqs):
63         probf[i] = freqs[i]/(r*c)
64
65     for i,j in enumerate(probf):
66         for k in range(i):
67             cum[i] += probf[k]
68     return cum
69
70 def myHE(input_file,cmap="gray"):
71     """
72     This is the Histogram Equalization Function.
73     input : the input image
74     output : None
75     Saves Histogram Equalized image
76     """
77     ## SETTING FONT-SIZE FOR PLOTTING
78     parameters = {'axes.titlesize': 10}
79     plt.rcParams.update(parameters)
80
81     name = input_file.split(".")[2]
82     input_image = cv2.imread(input_file)

```

```

83
84     new_image = np.zeros_like(input_image)
85
86     d = 1
87     if len(input_image.shape)>2:
88         r,c,d = input_image.shape
89     else:
90         r,c = input_image.shape
91
92     if d==1:
93         new_input = input_image
94         maximum = int(np.max(new_input))
95         cum = calculate_CDF(new_input,maximum,r,c)
96
97         for i in tqdm(range(r)):
98             for j in range(c):
99                 new_image[i,j] = truncateHE(cum[int(new_input[i][j])]*maximum)
100         plot_hist(input_file,input_image,new_image)
101
102     else:
103         input_image = cv2.cvtColor(input_image,cv2.COLOR_BGR2RGB)
104         hsv_image = cv2.cvtColor(input_image,cv2.COLOR_RGB2HSV)
105         output_image = zeros_like(hsv_image)
106         h,s,v = cv2.split(hsv_image)
107         v_copy = v.copy()
108         maximum = int(np.max(v))
109         cum = calculate_CDF(v,maximum,r,c)
110         for i in tqdm(range(r)):
111             for j in range(c):
112                 v_copy[i,j] = truncateHE(cum[int(v[i,j])]*maximum)
113
114         plot_hist(input_file,input_image,v_copy)
115
116         output_image[:, :, 2] = v_copy
117         output_image[:, :, 0] = h
118         output_image[:, :, 1] = s
119
120         output_image = cv2.cvtColor(output_image,cv2.COLOR_HSV2RGB)
121
122
123     fig,axes = plt.subplots(1,2, constrained_layout=True)
124
125
126     axes[0].imshow(input_image,cmap=cmap)
127     axes[0].axis("on")
128     axes[0].set_title("Original Image")
129
130     im = axes[1].imshow(output_image, cmap=cmap)
131     axes[1].axis("on")
132     axes[1].set_title("Histogram Equalized")
133
134     cbar = fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.35)
135     plt.savefig(".." + name + "HistEq.png",bbox_inches="tight",pad=-1)
136
137     if d==3:
138         plt.imsave(".." + name + "HE.png",output_image)
139     else:
140         plt.imsave(".." + name + "HE.png",output_image,cmap=cmap)

```

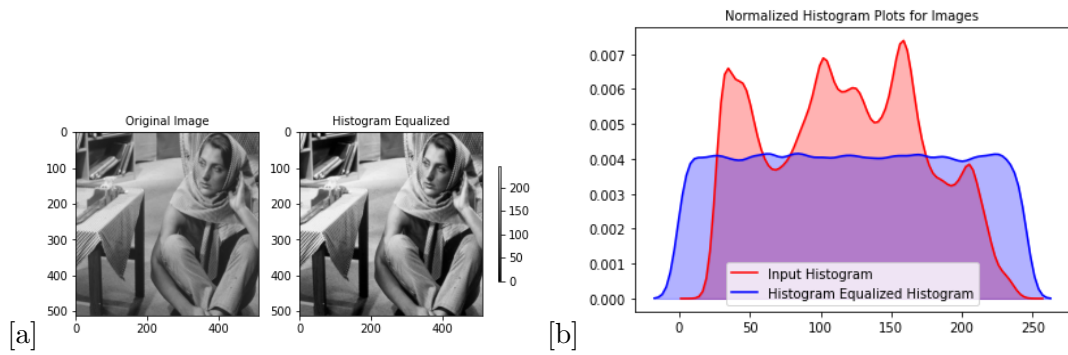



Figure 8: (a) HE for barbara.png (b) Histogram comparison for barbara.png

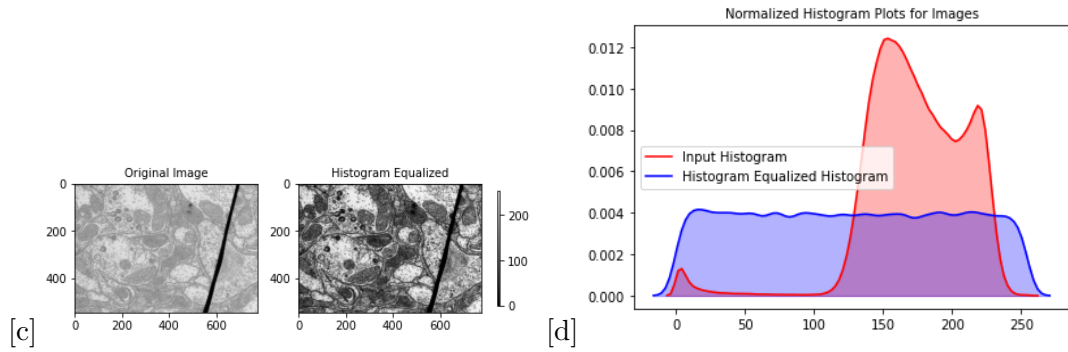


Figure 9: (c) HE for TEM.png (d) Histogram comparison for TEM.png

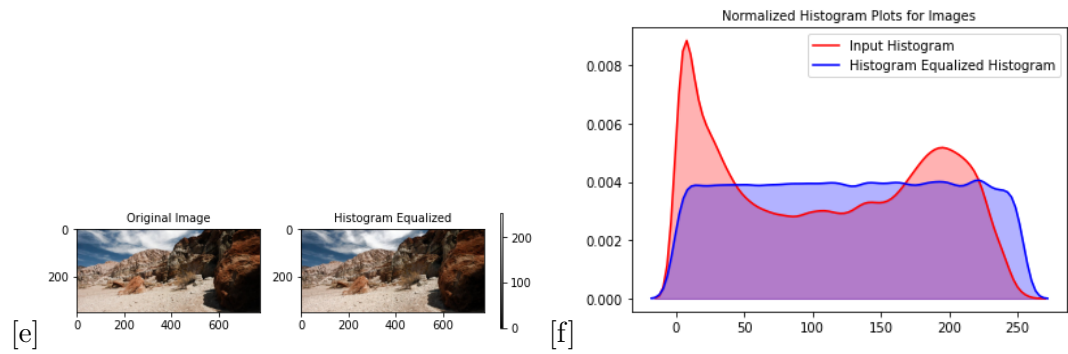


Figure 10: (e) HE for canyon.png (f) Histogram comparison for canyon.png

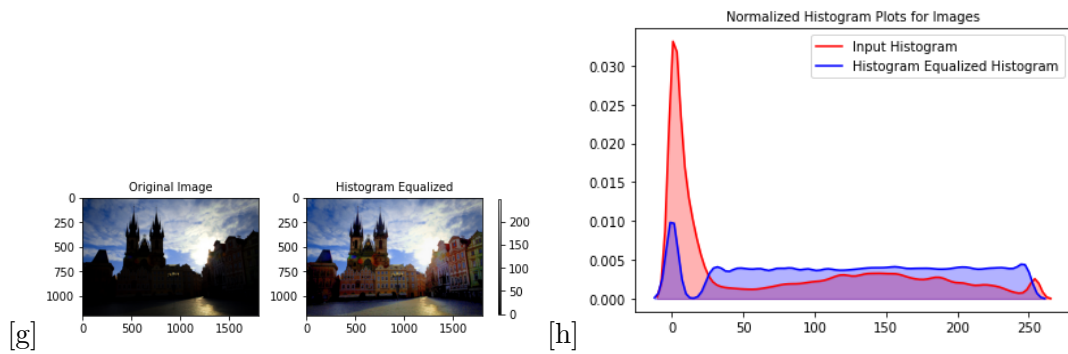


Figure 11: (g) HE for church.png (f) Histogram comparison for church.png

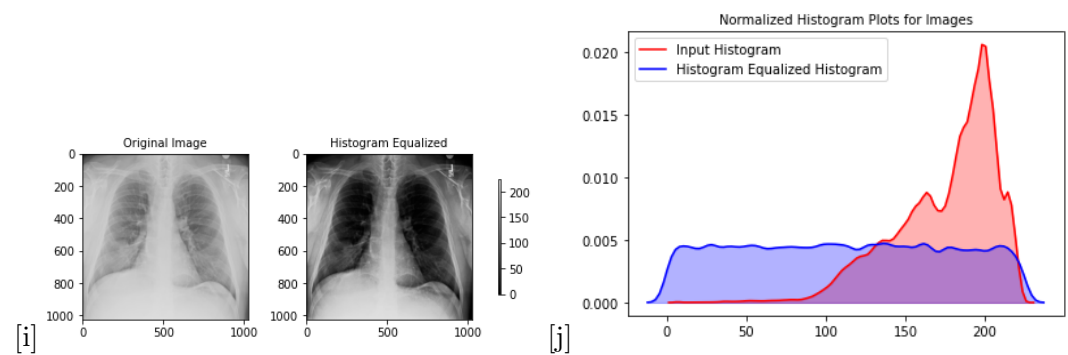


Figure 12: (i) HE for chestXray.png (j) Histogram comparison for chestXray.png

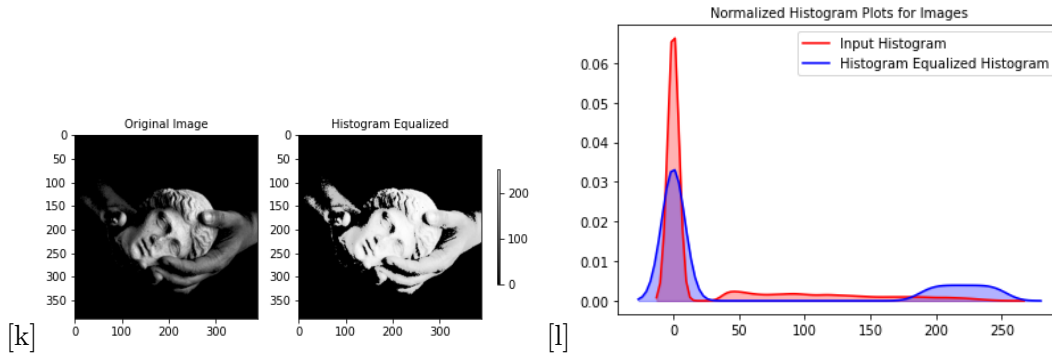


Figure 13: (k) HE for statueForegroundMasked.png (l) Histogram comparison for statueForegroundMasked.png

Observation for image (5) church.png The following are the histograms of *church.png* after LCS and HE.

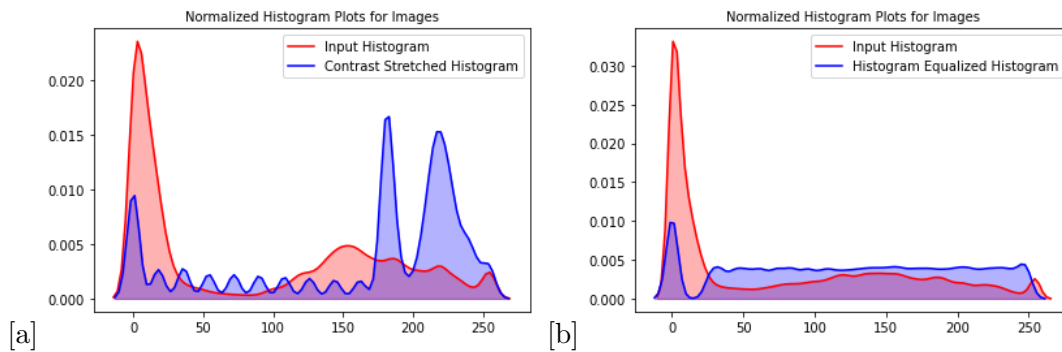


Figure 14: (a) Histogram after LCS (b) Histogram after HE

Observation :

We observe that in Histogram Equalization, the Histogram obtained is more or less flat along all the pixel intensities, while that in case of Linear Contrast Stretching (LCS) the pixel intensities seem to be concentrated only in two major intensity ranges. So, Histogram Equalization of image *church.png* would be preferred to LCS as the output of the HE image is of better contrast than LCS.

In general, we have that Histogram Equalization works better when the histogram of an image is confined to a region in the pixel intensities, which is the case of *church.png*.

2.4 (15 points) Histogram Matching (HM)

You are provided two images: input image *2/data/retina.png* and reference image *2/data/retinaRef.png*. Perform HM on the input image by matching the histogram with that of the reference image. Note that you don't need to show results for other images for this part of the question.

- Write a function *myHM.m* to implement this.
- Display the original image, histogram-matched image and the histogram-equalised image.
- State your observations.

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3  import matplotlib.image as mpimg
4  import matplotlib as mpl
5
6  from tqdm import tqdm
7  import cv2
8  from seaborn import distplot
9  from math import floor,ceil
10
11 def plot_hist(input_file,input_image,output_image,reference_image):
12     name = input_file.split(".")[2]
13     plt.figure()
14     plt.title("Normalized Histogram Plots for Images")
15     ax = distplot(input_image,color='r',label ="Input Histogram",

```

```

16     hist_kws={"alpha": 0.3, "linewidth": 1.5},bins=256,hist=False)
17     ax = distplot(output_image,color="b",label ="Histogram Matched Histogram",
18     hist_kws={"alpha": 0.3,"linewidth": 1.5},bins=256,hist=False)
19     ax = distplot(reference_image,color='g',label ="Reference Histogram",
20     hist_kws={"alpha": 0.3, "linewidth": 1.5},bins=256,hist=False)
21
22     l1 = ax.lines[0]
23     x1 = l1.get_xydata()[:,0]
24     y1 = l1.get_xydata()[:,1]
25     ax.fill_between(x1,y1, color="red", alpha=0.3)
26     l2 = ax.lines[1]
27     x2 = l2.get_xydata()[:,0]
28     y2 = l2.get_xydata()[:,1]
29     ax.fill_between(x2,y2, color="blue", alpha=0.3)
30     l3 = ax.lines[2]
31     x3 = l3.get_xydata()[:,0]
32     y3 = l3.get_xydata()[:,1]
33     ax.fill_between(x3,y3, color="green", alpha=0.3)
34     plt.legend()
35     plt.savefig("../"+name+"HMHistogram.png",bbox_inches="tight",pad=-1)
36
37
38 def perform_masking(original,masking,r,c,d=3):
39     """
40     Masks the original image using the mask provided
41     input : orig_image, mask, r(rows of orig_image), c(cols of orig_image),
42           d(#channels of orig_image)
43     output : masked image
44     """
45     orig = original.copy()
46     mask = masking.copy()
47
48     for i in range(3):
49         for j in range(r):
50             for k in range(c):
51                 orig[j,k,i] = (0 if mask[j,k,i]==0 else orig[j,k,i])
52
53     return orig
54
55 def calculate_CDF(array,maximum,r,c):
56     """
57     This function is used to calculate the CDF of the 2D image.
58     array : For gray scale the whole image is the input and for RGB
59           each of the color slices are the inputs.
60     maximum : maximum pixel intensity of the 2D image
61     output : the CDF of the 2D image
62     """
63     freqs = np.zeros((maximum+1,1))
64     probf = np.zeros((maximum+1,1))
65     cum = np.zeros((maximum+1,1))
66
67     for i in range(r):
68         for j in range(c):
69             freqs[int(array[i][j])]+=1
70
71     for i,j in enumerate(freqs):
72         probf[i] = freqs[i]/(r*c)
73
74     for i,j in enumerate(probf):
75         for k in range(i):
76             cum[i] += probf[k]
77     return cum
78
79 def myHM(reference,reference_mask,target,target_mask):
80

```

```

81 parameters = {'axes.titlesize': 10}
82 plt.rcParams.update(parameters)
83
84 name = target.split(".")[2]
85
86 parameters = {'axes.titlesize': 10}
87 plt.rcParams.update(parameters)
88
89 # READING THE REFERENCE IMAGE
90 original_ref_image = cv2.imread(reference)
91 original_ref_image = cv2.cvtColor(original_ref_image,cv2.COLOR_BGR2RGB)
92
93 r1,c1,d1 = original_ref_image.shape
94 # READING THE REFERENCE IMAGE MASK AND PRODUCING THE MASKED REFERENCE IMAGE
95 original_ref_image_mask = cv2.imread(reference_mask)
96 original_ref_image_mask = cv2.cvtColor(original_ref_image_mask,cv2.COLOR_BGR2RGB)
97 original_ref_masked = perform_masking(original_ref_image,original_ref_image_mask,r1,c1,d1)
98
99 # READING THE IMAGE TO BE HISTOGRAM MATCHED
100 original_target_image = cv2.imread(target)
101 original_target_image = cv2.cvtColor(original_target_image,cv2.COLOR_BGR2RGB)
102
103 r2,c2,d2 = original_target_image.shape
104
105 # READING THE MASK OF THE IMAGE TO BE MATCHED
106 original_target_image_mask = cv2.imread(target_mask)
107 original_target_image_mask = cv2.cvtColor(original_target_image_mask,cv2.COLOR_BGR2RGB)
108 original_target_masked = perform_masking(original_target_image,original_target_image_mask,r2,c2,d2)
109
110 # CREATING THE HSV TRANSFORM FOR THE MASKED IMAGES AND EXTRACTING THE V-CHANNEL
111 ref_hsv = cv2.cvtColor(original_ref_masked,cv2.COLOR_RGB2HSV)
112 ref_v = cv2.split(ref_hsv)[2]
113 target_hsv = cv2.cvtColor(original_target_masked,cv2.COLOR_RGB2HSV)
114 target_v = cv2.split(target_hsv)[2]
115
116 ref_image = ref_hsv.copy()
117 target_image = target_hsv.copy()
118
119 # CUMULATIVE DISTRIBUTION OF THE IMAGES
120 cum_ref = calculate_CDF(ref_v,int(np.max(ref_v)),r1,c1)
121 cum_target = calculate_CDF(target_v,int(np.max(target_v)),r2,c2)
122
123 transform_ref=[0 for i in range(256)]
124 transform_tar=[0 for i in range(256)]
125
126 for i in range(len(cum_target)):
127     transform_tar[i] = floor(255*cum_target[i])
128     transform_ref[i] = floor(255*cum_ref[i])
129
130 transform ={}
131
132 for i in transform_tar:
133     value=min(transform_ref, key=lambda x:abs(x-i))
134     indx = transform_ref.index(value)
135     original = transform_tar.index(i)
136     transform[original]=indx
137
138 for i in range(r1):
139     for j in range(c1):
140         if(target_image[i,j,2] in transform):
141             target_image[i,j,2]=transform[target_image[i,j,2]]
142
143 plot_hist(target,target_v,target_image[:, :, 2],ref_v)
144
145 # CONVERT BACK FROM HSV TO RGB

```

```

146 target_image = cv2.cvtColor(target_image,cv2.COLOR_HSV2RGB)
147
148
149
150 fig,axes = plt.subplots(1,3, constrained_layout=True)
151
152 axes[0].imshow(original_ref_masked)
153 axes[0].axis("on")
154 axes[0].set_title("Reference Image")
155 axes[1].imshow(original_target_masked)
156 axes[1].axis("on")
157 axes[1].set_title("Original Image")
158 im = axes[2].imshow(target_image)
159 axes[2].axis("on")
160 axes[2].set_title("Histogram Matched")
161 cbar = fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.45)
162 plt.savefig("../"+name+"HistogramMatched.png",cmap="gray",bbox_inches="tight",pad=-1)
163
164 plt.imsave("../" + name+"HM",target_image,cmap="gray")

```

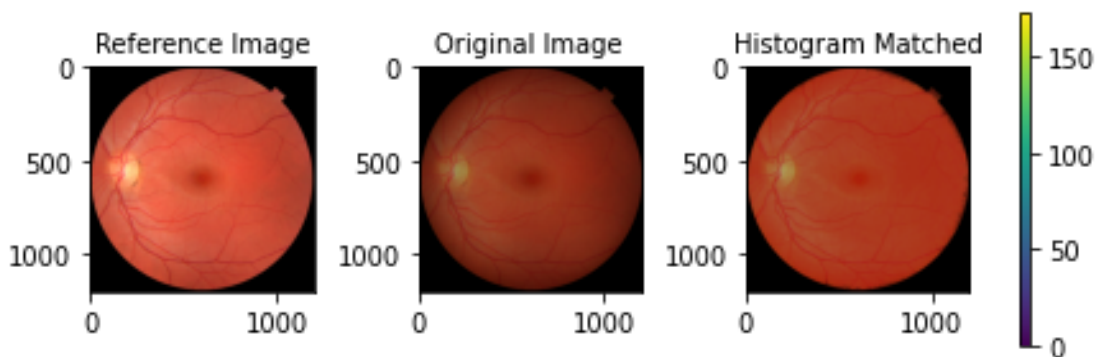


Figure 15: Output of myHM.py

Observations :

We see that the Original retina image was quite dark while the Reference image was quite bright. So, on Histogram matching the output image was pushed to have the similar histogram pattern to that of the Reference. We observe that, although the image brightens a bit, but still the arteries are not quite visible as that in the Reference image.

2.5 (30 points) Contrast-Limited Adaptive Histogram Equalization (CLAHE)

Perform CLAHE on the entire image. Manually tune the window-size parameter and the histogram-threshold parameter $\in [0, 1]$ to an appropriate values in order to perform contrast enhancement for objects in the image, while preventing excessive noise amplification. For pixels close to the boundary, where the window falls outside the image, use only the pixels that are within the window and within the image (i.e, effectively cropping the window to restrict it within the image boundaries).

- Write a function *myCLAHE.m* to implement this.
- Display the original image and the contrast-enhanced image. Redo CLAHE with neighborhood sizes chosen to be
 1. significantly larger
 2. significantly smaller than that chosen for the previous result in order to clearly show the effects of (i) low contrast improvement and (ii) excessive noise amplification
- Display the additional two contrast-enhanced images.Redo CLAHE with
 1. the same window size as before
 2. histogram-threshold parameter set to a value that is half of the value tuned before
- Display the additional contrast-enhanced images.
- Show the above results on input images 1, 2, 3, 6.

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3  import matplotlib.image as mpimg
4  from numpy import zeros_like
5
6  from tqdm import tqdm
7  from math import floor
8  import cv2
9  from seaborn import distplot
10
11 def plot_hist(input_file,input_image,output_image>window_x,threshold):
12     """
13     input : input_file_path, input_image, output_image>window_size, histogram threshold
14     output : saves the histograms for both the images for comparison
15     dependencies : seaborn, numpy, matplotlib
16     """
17     name = input_file.split(".")[2]
18     plt.figure()
19     plt.title("Normalized Histogram Plots for Images")
20     ax = distplot(input_image,color='r',label = "Input Histogram",
21         hist_kws={"alpha": 0.3, "linewidth": 1.5},bins=256,hist=False)
22     ax = distplot(output_image,color="b",label = "CLAHE Histogram",
23         hist_kws={"alpha": 0.3,"linewidth": 1.5},bins=256,hist=False)
24     l1 = ax.lines[0]
25     x1 = l1.get_xydata()[:,0]
26     y1 = l1.get_xydata()[:,1]
27     ax.fill_between(x1,y1, color="red", alpha=0.3)
28     l2 = ax.lines[1]
29     x2 = l2.get_xydata()[:,0]
30     y2 = l2.get_xydata()[:,1]
31     ax.fill_between(x2,y2, color="blue", alpha=0.3)
32     plt.legend()
33     plt.savefig("."+name+"CLAHEHistogram_"+str(window_x*2)+"_"+str(threshold)+".png",
34         bbox_inches="tight",pad=-1)
35
36 def imhist(input_array):
37     m, n = input_array.shape
38     h = [0.0] * 256
39     for i in range(m):
40         for j in range(n):
41             h[int(input_array[i, j])]+=1
42     return np.array(h)/(m*n)
43
44 def calcCLAHEVal(input_array,th):
45     r,c = input_array.shape
46     H = imhist(input_array)
47     C = zeros_like(H)
48     for i in range(256):
49         if H[i] > th:
50             H[i] = th
51     contrastArea = 1 - sum(H)
52     height = contrastArea/256
53     H = H + height
54     C[0] = H[0]
55
56     for i in range(1,256):
57         C[i] = C[i-1] + (H[i])
58
59     return C
60
61 def myCLAHE(input_file>window_x>window_y,threshold,cmap):
62
63     parameters = {'axes.titlesize': 10}
64     plt.rcParams.update(parameters)

```

```

65
66     name = input_file.split(".")[2]
67     input_image = cv2.imread(input_file)
68     output_image = np.zeros_like(input_image)
69     d=1
70     if len(input_image.shape)<3:
71         r,c = input_image.shape
72     else:
73         r,c,d = input_image.shape
74
75
76     if d==1:
77         new_image = input_image
78
79         for i in tqdm(range(r)):
80             for j in range(c):
81                 min_x = max(0,i-window_x)
82                 min_y = max(0,j-window_y)
83                 max_x = min(r,i+window_x)
84                 max_y = min(c,j+window_y)
85
86                 window_image = input_image[min_x:max_x,min_y:max_y]
87                 if new_image[i,j]!=0:
88                     x = calcCLAHEVal(window_image,threshold)
89                     output_image[i,j] = x[int(new_image[i,j])]*255
90
91                 plot_hist(input_file,input_image,output_image,window_x,threshold)
92
93     else:
94         output_hsv = np.zeros_like(input_image)
95         input_image = cv2.cvtColor(input_image,cv2.COLOR_BGR2RGB)
96         hsv_image = cv2.cvtColor(input_image, cv2.COLOR_RGB2HSV)
97         h,s,v = cv2.split(hsv_image)
98
99         out_v = v.copy()
100
101         for i in tqdm(range(r)):
102             for j in range(c):
103                 min_x = max(0,i-window_x)
104                 min_y = max(0,j-window_y)
105                 max_x = min(r,i+window_x)
106                 max_y = min(c,j+window_y)
107
108                 window_image = v[min_x:max_x,min_y:max_y]
109                 if out_v[i,j]!=0:
110                     x = calcCLAHEVal(window_image,threshold)
111                     out_v[i,j] = x[int(v[i,j])]*255
112
113                 plot_hist(input_file,v,out_v,window_x,threshold)
114
115         output_hsv[:, :, 0] = h
116         output_hsv[:, :, 1] = s
117         output_hsv[:, :, 2] = out_v
118
119         output_image = cv2.cvtColor(output_hsv,cv2.COLOR_HSV2RGB)
120
121     fig,axes = plt.subplots(1,2, constrained_layout=True)
122     axes[0].imshow(input_image,cmap=cmap)
123     axes[0].axis("on")
124     axes[0].set_title("Original Image")
125     im = axes[1].imshow(output_image,cmap=cmap)
126     axes[1].axis("on")
127     axes[1].set_title("CLAHE Image")
128     cbar = fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.45)
129

```



```

130 plt.savefig("../"+name+"CLAHEcombined_"+str(window_x*2)+"_"+str(threshold)+".png",
131             bbox_inches="tight",pad=-1)
132
133 plt.imshow(output_image,cmap=cmap)

```

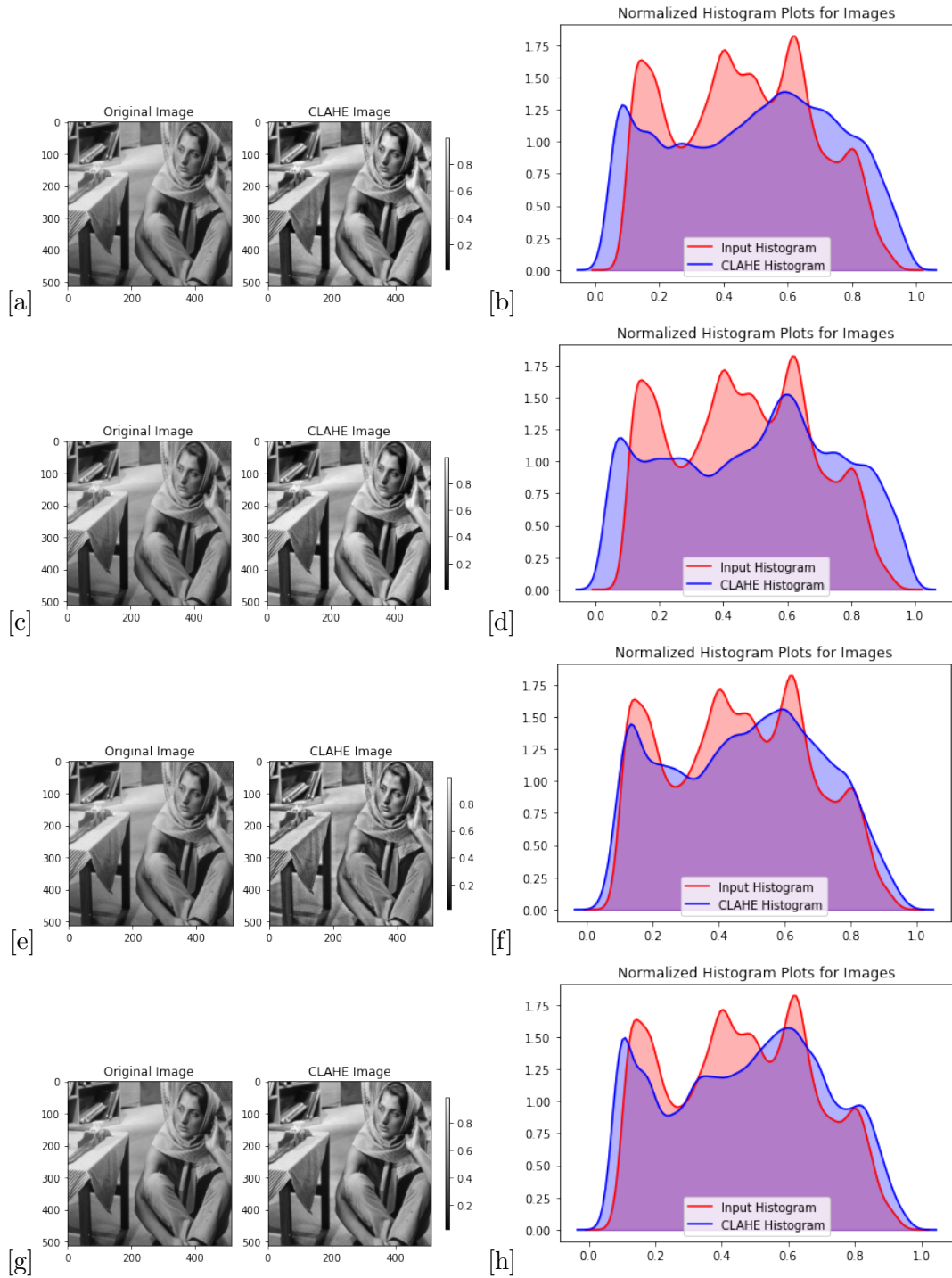


Figure 16: CLAHE operation on barbara.png with different Window Sizes and Threshold Values (a>window:128 th:0.005 (b) Histogram for (a) (c) window:256 th:0.005 (d) Histogram for (c) (e) window:32 th:0.005 (f) Histogram for (e) (g) window:128 th:0.0025 (h) Histogram for (g)

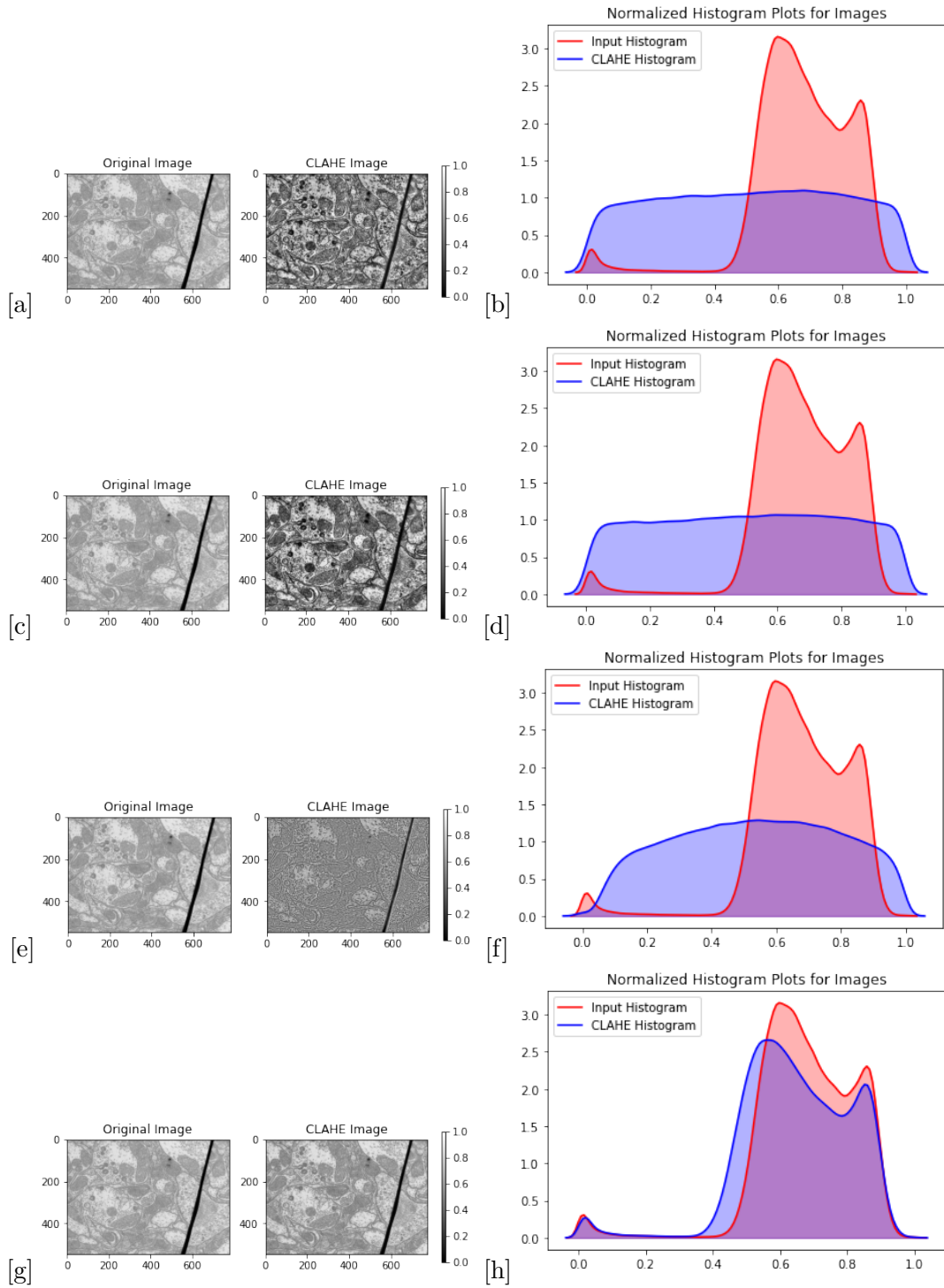


Figure 17: CLAHE operation on TEM.png with different Window Sizes and Threshold Values (a>window:64 th:0.03 (b) Histogram for (a) (c) window:128 th:0.03 (d) Histogram for (c) (e) window:8 th:0.03 (f) Histogram for (e) (g) window:64 th:0.015 (h) Histogram for (g)

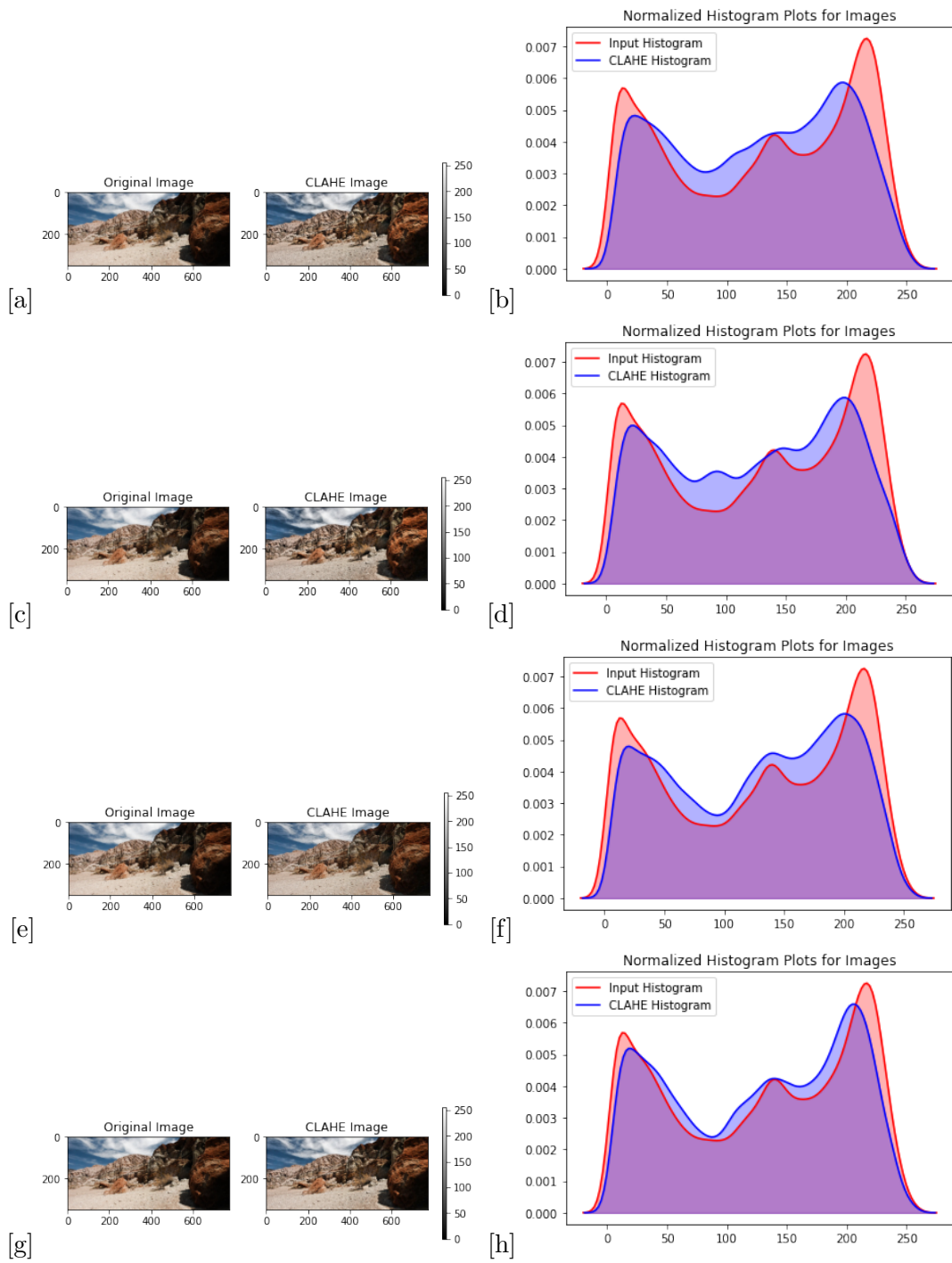


Figure 18: CLAHE operation on canyon.png with different Window Sizes and Threshold Values (a>window:32 th:0.005 (b) Histogram for (a) (c>window:64 th:0.005 (d) Histogram for (c) (e) window:8 th:0.005 (f) Histogram for (e) (g) window:32 th:0.0025 (h) Histogram for (g)

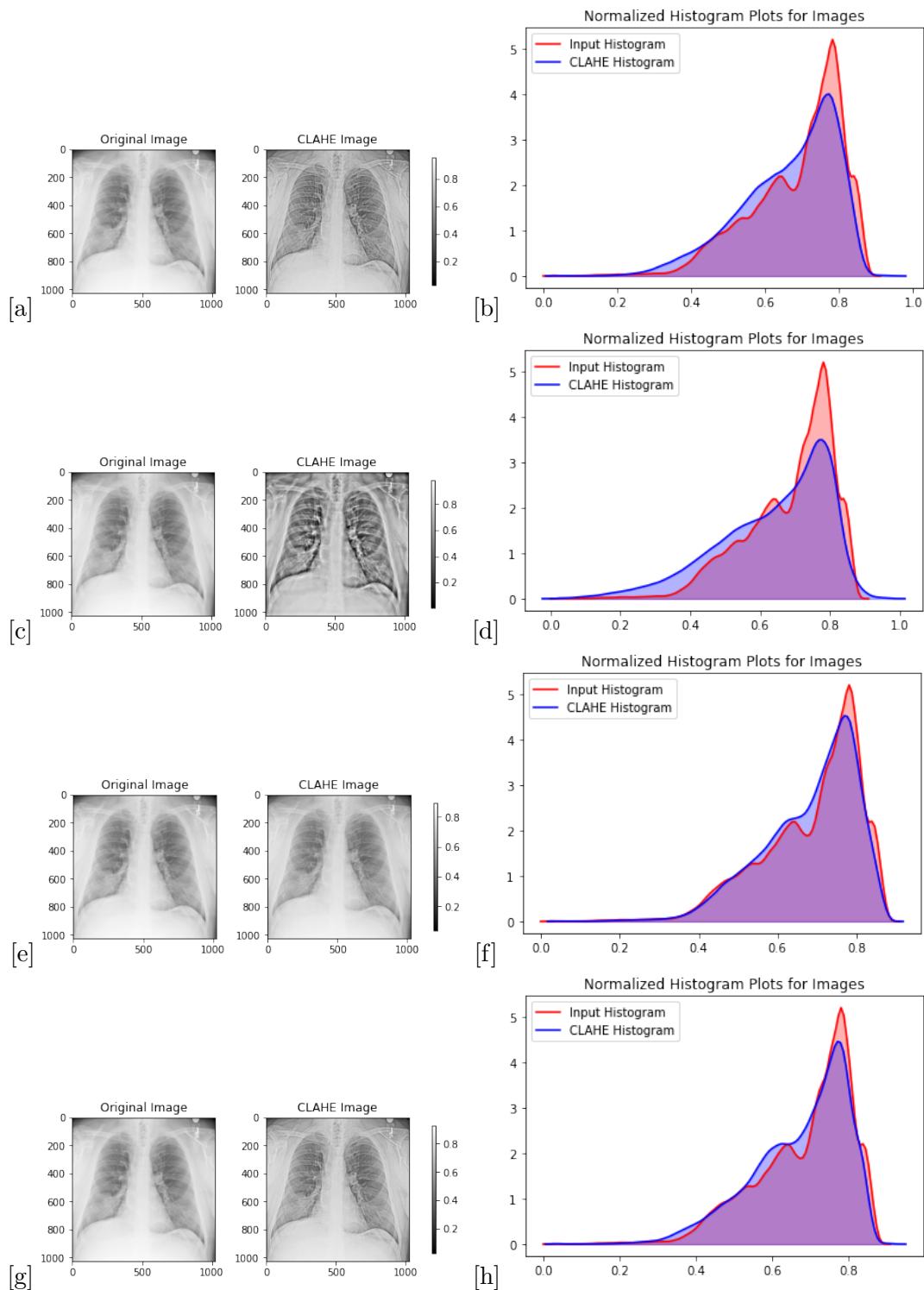


Figure 19: CLAHE operation on chestXray.png with different Window Sizes and Threshold Values (a>window:16 th:0.015 (b) Histogram for (a) (c) window:64 th:0.015 (d) Histogram for (c) (e) window:4 th:0.015 (f) Histogram for (e) (g) window:16 th:0.0075 (h) Histogram for (g)

2.6 8

myMainScript.py

```

1  from myHM import myHM
2  from myHE import myHE
3  from myForegroundMask import myForegroundMask
4  from myCLAHE import myCLAHE
5  from myLinearContrastStretching import myLinearContrastStretching
6
7  from time import time
8
9  image1 = "../data/barbara.png"
10 image2 = "../data/TEM.png"
11 image3 = "../data/canyon.png"
12 image4 = "../data/retina.png"
13 image5 = "../data/church.png"
14 image6 = "../data/chestXray.png"

```

```

15 image7 = "../data/statue.png"
16
17 maskedStatue = "../data/statueForegroundMasked.png"
18 reference = "../data/retinaRef.png"
19 reference_mask = "../data/retinaRefMask.png"
20 target = "../data/retina.png"
21 target_mask = "../data/retinaMask.png"
22
23 superStart = time()
24
25 start = time()
26 myForegroundMask(image7)
27 end = time()
28 print("Time to run myForegroundMask.py :",end-start,"secs")
29
30 start = time()
31 myLinearContrastStretching(image1,[120,230],[90,245])
32 myLinearContrastStretching(image2,[120,230],[30,245])
33 myLinearContrastStretching(image3,[100,180],[60,200])
34 myLinearContrastStretching(image5,[10,200],[180,230])
35 myLinearContrastStretching(image6,[100,200],[30,180])
36 myLinearContrastStretching(maskedStatue,[50,200],[20,230])
37 end = time()
38 print("Time to run myLinearContrastStretching.py :",end-start,"secs")
39
40
41 start = time()
42 myHE(image1)
43 myHE(image2)
44 myHE(image3)
45 myHE(image5)
46 myHE(image6)
47 myHE(maskedStatue)
48 end = time()
49 print("Time to run myHE.py :",end-start,"secs")
50
51 start = time()
52 myHM(reference,reference_mask,target,target_mask)
53 end = time()
54 print("Time to run myHM.py :",end-start,"secs")
55
56 start = time()
57 myCLAHE(image1,64,64,0.005,"gray")
58 myCLAHE(image1,128,128,0.005,"gray")
59 myCLAHE(image1,16,16,0.005,"gray")
60 myCLAHE(image1,64,64,0.0025,"gray")
61 myCLAHE(image2,32,32,0.03,"gray")
62 myCLAHE(image2,64,64,0.03,"gray")
63 myCLAHE(image2,4,4,0.03,"gray")
64 myCLAHE(image2,32,32,0.0015,"gray")
65 myCLAHE(image3,16,16,0.005,"gray")
66 myCLAHE(image3,32,32,0.005,"gray")
67 myCLAHE(image3,8,8,0.005,"gray")
68 myCLAHE(image3,16,16,0.0025,"gray")
69 myCLAHE(image6,8,8,0.015,"gray")
70 myCLAHE(image6,32,32,0.015,"gray")
71 myCLAHE(image6,2,2,0.015,"gray")
72 myCLAHE(image6,8,8,0.0075,"gray")
73 myCLAHE(maskedStatue)
74 end = time()
75 print("Time to run myCLAHE.py :",end-start,"secs")
76
77 superEnd = time()
78 print("Time required to run codes for Q2 :",round(superEnd-superStart,2),"minutes")

```