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.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import matplotlib as mpl
import cv2


def myForegroundMask(input_file,cmap="gray",offset=2):
    name = input_file.split(".")[2]
    input_image = cv2.imread(input_file,0)
    new_image = np.zeros_like(input_image)

    # PLOTTING PARAMETERS
    parameters = {'axes.titlesize': 10}
    plt.rcParams.update(parameters)

    print(np.max(input_image))

    r,c = input_image.shape

```

```python
    # mask formation
    for i in range(input_image.shape[0]):
        for j in range(input_image.shape[1]):
            if input_image[i][j] < np.mean(input_image)+offset:
                new_image[i][j]=0
            else:
                new_image[i][j]=1

    # masked image formation
    masked_image = new_image*input_image


    fig,axes = plt.subplots(1,3, constrained_layout=True)

    axes[0].imshow(input_image,cmap="gray")
    axes[0].axis("on")
    axes[0].set_title("Original Image")

    axes[1].imshow(new_image,cmap="gray")
    axes[1].axis("on")
    axes[1].set_title("Foreground Mask(th = 2)")

    im = axes[2].imshow(masked_image*2,cmap="gray")
    axes[2].axis("on")
    axes[2].set_title("Masked Image")

    fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.45)
    plt.savefig(".."+name+"ForegroundMask.png",cmap=cmap,bbox_inches="tight",pad=-1)

    cv2.imwrite(".." + name+"Mask.png",new_image)
    cv2.imwrite(".." + name+"ForegroundMasked.png",masked_image)
```

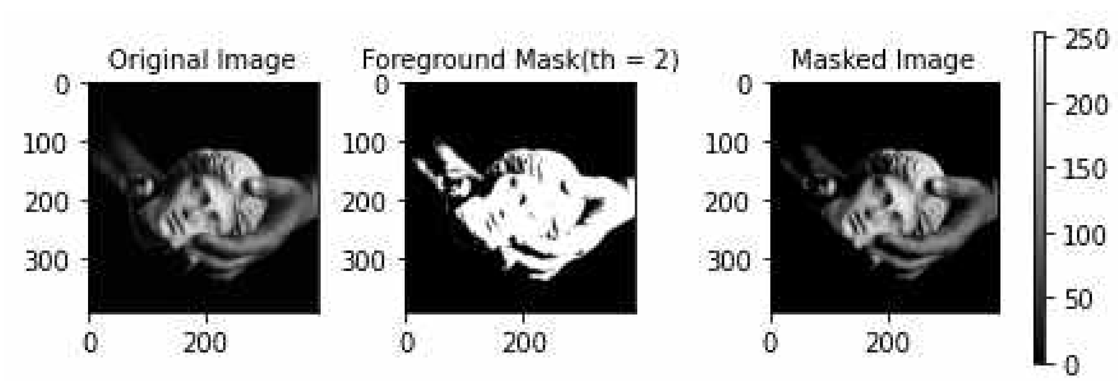


Figure 1: Output of myForegroundMask.py

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?

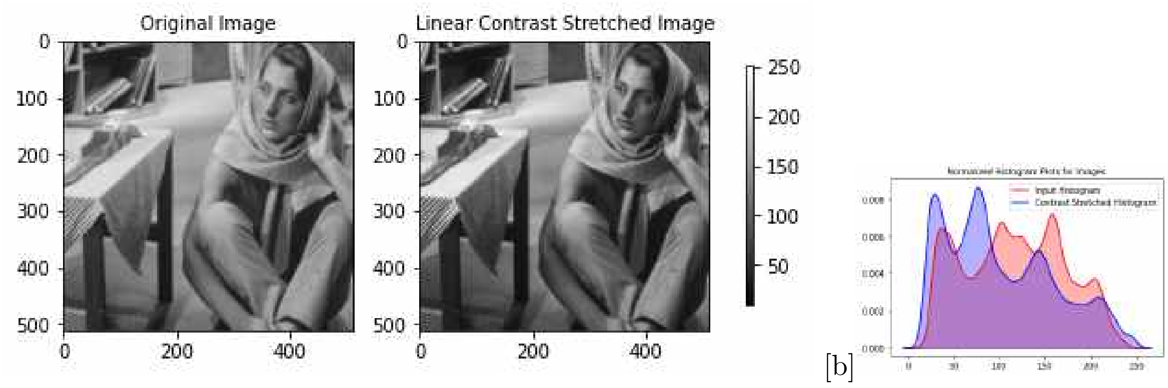
Formula used for Linear Contrast Stretching:

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

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.

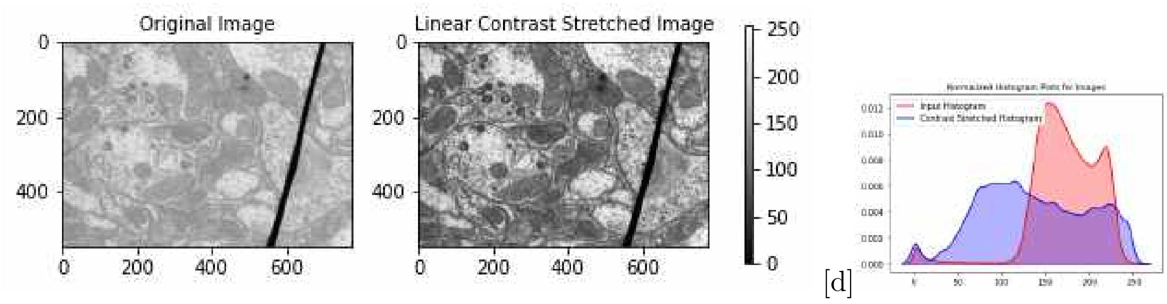
```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import matplotlib as mpl

from tqdm import tqdm
import cv2
from seaborn import distplot

def plot_hist(input_file,input_image,output_image):
    """
    input : input_file_path, input_image, output_image
    output : saves the histograms for both the images for comparison
    dependencies : seaborn, numpy, matplotlib
    """
    name = input_file.split(".")[2]
    plt.figure()
    plt.title("Normalized Histogram Plots for Images")
    ax = distplot(input_image,color='r',label ="Input Histogram",
        hist_kws={"alpha": 0.3, "linewidth": 1.5},bins=256,hist=False)
    ax = distplot(output_image,color="b",label ="Contrast Stretched Histogram",
        hist_kws={"alpha": 0.3,"linewidth": 1.5},bins=256,hist=False)
    l1 = ax.lines[0]
    x1 = l1.get_xydata()[:,0]
    y1 = l1.get_xydata()[:,1]
    ax.fill_between(x1,y1, color="red", alpha=0.3)
    l2 = ax.lines[1]
    x2 = l2.get_xydata()[:,0]
    y2 = l2.get_xydata()[:,1]
    ax.fill_between(x2,y2, color="blue", alpha=0.3)
    plt.legend()
    plt.savefig(".."+name+"LCSHistogram.png",bbox_inches="tight",pad=-1)


def truncate(array):
    """
    input : array
    output : truncated array to make it stay from 0 to 255
    """
    r,c = array.shape
    for i in range(r):
        for j in range(c):
            if array[i][j]<0.0:
                array[i][j] = 0
            elif array[i][j]>255.0:
                array[i][j]=255.0
    return array


def myLinearContrastStretching(input_file,x1=[0,255],x2=[0,255],cmap="gray"):
    """
    input : <input_file_path>, input_image_range(x1), output_image_range(x2), cmap(optional)
    output : Saves the linear contrast stretched image
    x1 : [r1,r2] (by default = [0,255])
    x2 : [s1,s2] (by default = [0,255])
    """
```

```
    parameters = {'axes.titlesize': 10}
    plt.rcParams.update(parameters)

    r1,r2 = x1
    s1,s2 = x2
    name = input_file.split(".")[2]
    input_image = cv2.imread(input_file)
    d = 1
    if len(input_image.shape)>2:
        r,c,d = input_image.shape
    else:
        r,c = input_image.shape

    if d==1:
        new_image=np.zeros_like(input_image)
        for i in tqdm(range(r)):
            for j in range(c):
                input_pixel = input_image[i,j]
                if input_pixel<=r1:
                    input_pixel = (s1/r1) * input_pixel
                elif input_pixel>r1 and input_pixel <= r2:
                    input_pixel = ((s2-s1)/(r2-r1))*(input_pixel-r1) + s1
                else:
                    input_pixel = ((255.0-s2)/(255.0-r2))*(input_pixel-r2) + s2
                new_image[i][j]= input_pixel
        new_image = truncate(new_image)
        plot_hist(input_file,input_image,new_image)

    else:
        input_image = cv2.cvtColor(input_image,cv2.COLOR_BGR2RGB)
        hsv_image = cv2.cvtColor(input_image,cv2.COLOR_RGB2HSV)
        h,s,v = cv2.split(hsv_image)
        v_new = v.copy()
        for i in tqdm(range(r)):
            for j in range(c):
                input_pixel = v[i,j]
                if input_pixel<=r1:
                    input_pixel = (s1/r1) * input_pixel
                elif input_pixel>r1 and input_pixel <= r2:
                    input_pixel = ((s2-s1)/(r2-r1))*(input_pixel-r1) + s1
                else:
                    input_pixel = ((255.0-s2)/(255.0-r2))*(input_pixel-r2) + s2
                v_new[i,j]= input_pixel

        hsv_image[:,:,2] = v_new
        plot_hist(input_file,v,v_new)
        new_image = cv2.cvtColor(hsv_image,cv2.COLOR_HSV2RGB)


    fig,axes = plt.subplots(1,2, constrained_layout=True)
    axes[0].imshow(input_image,cmap="gray")
    axes[0].axis("on")
    axes[0].set_title(r"Original Image")
    im = axes[1].imshow(new_image,cmap="gray")
    axes[1].axis("on")
    axes[1].set_title(r"Linear Contrast Stretched Image")

    cbar = fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.45)
    plt.savefig(".."+name+"LCS.png",bbox_inches="tight",pad=-1)

    plt.imsave(".." + name+"LinearContrastStretching.png",new_image,cmap=cmap)
```



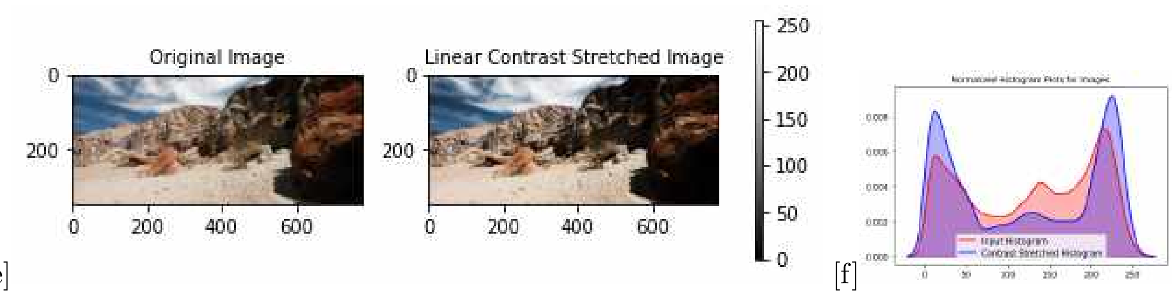
[a] [b]

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



[c] [d]

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

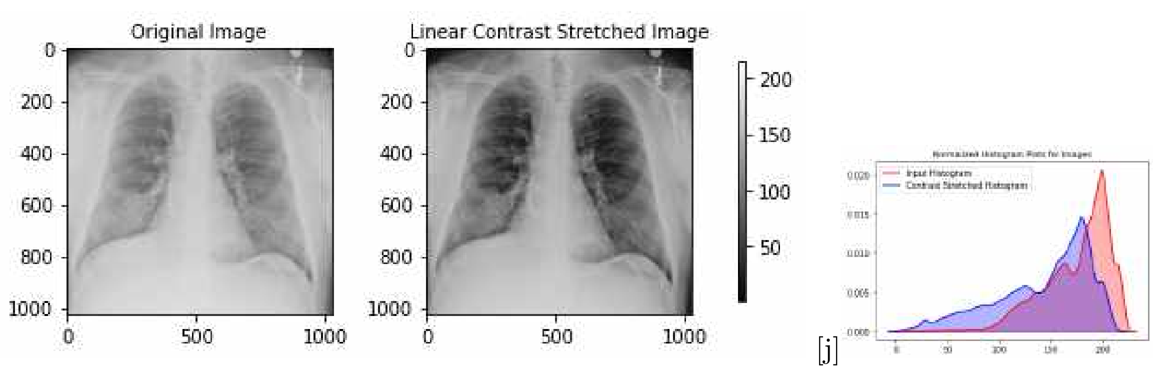


[e] [f]

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

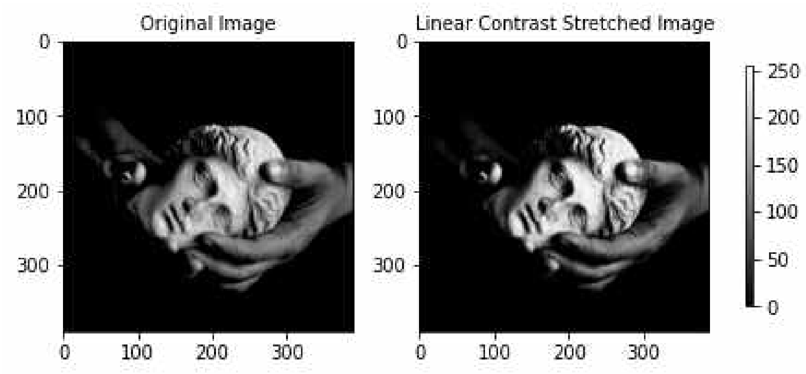
[g] [h]

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



[i] [j]

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

[k] 

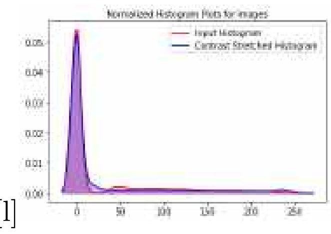
[l] 

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?

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from numpy import zeros_like
from tqdm import tqdm
from math import floor
import cv2
from seaborn import distplot

def plot_hist(input_file,input_image,output_image):
    """
    input : input_file_path, input_image, output_image
    output : saves the histograms for both the images for comparison
    dependencies : seaborn, numpy, matplotlib
    """
    name = input_file.split(".")[2]
    plt.figure()
    plt.title("Normalized Histogram Plots for Images")
    ax = distplot(input_image,color='r',label ="Input Histogram",hist_kws={"alpha": 0.3,
        "linewidth": 1.5},bins=256,hist=False)
    ax = distplot(output_image,color="b",label ="Histogram Equalized Histogram",
        hist_kws={"alpha": 0.3,"linewidth": 1.5},bins=256,hist=False)
    l1 = ax.lines[0]
    x1 = l1.get_xydata()[:,0]
    y1 = l1.get_xydata()[:,1]
    ax.fill_between(x1,y1, color="red", alpha=0.3)
    l2 = ax.lines[1]
    x2 = l2.get_xydata()[:,0]
    y2 = l2.get_xydata()[:,1]
    ax.fill_between(x2,y2, color="blue", alpha=0.3)
    plt.legend()
```

```
    plt.savefig(".."+name+"HEHistogram.png",bbox_inches="tight",pad=-1)


def truncateHE(array):
    """
    This function truncates the array values to check whether the values
    are within range of [0,255]
    """
    if array<0:
        array = 0
    elif array>255.0:
        array = 255.0
    return array

def calculate_CDF(array,maximum,r,c):
    """
    This function is used to calculate the CDF of the 2D image.
    array : For gray scale the whole image is the input and for RGB each
    of the color slices are the inputs.
    maximum : maximum pixel intensity of the 2D image
    output : the CDF of the 2D image
    """
    freqs = np.zeros((maximum+1,1))
    probf = np.zeros((maximum+1,1))
    cum = np.zeros((maximum+1,1))

    for i in range(r):
        for j in range(c):
            freqs[int(array[i][j])]+=1

    for i,j in enumerate(freqs):
        probf[i] = freqs[i]/(r*c)

    for i,j in enumerate(probf):
        for k in range(i):
            cum[i] += probf[k]
    return cum

def myHE(input_file,cmap="gray"):
    """
    This is the Histogram Equalization Function.
    input : the input image
    output : None
    Saves Histogram Equalized image
    """
    ## SETTING FONT-SIZE FOR PLOTTING
    parameters = {'axes.titlesize': 10}
    plt.rcParams.update(parameters)

    name = input_file.split(".")[2]
    input_image = cv2.imread(input_file)

    new_image = np.zeros_like(input_image)

    d = 1
    if len(input_image.shape)>2:
        r,c,d = input_image.shape
    else:
        r,c = input_image.shape

    if d==1:
        new_input = input_image
        maximum = int(np.max(new_input))
        cum = calculate_CDF(new_input,maximum,r,c)

```

```
        for i in tqdm(range(r)):
            for j in range(c):
                new_image[i,j] = truncateHE(cum[int(new_input[i][j])]*maximum)
        plot_hist(input_file,input_image,new_image)

    else:
        input_image = cv2.cvtColor(input_image,cv2.COLOR_BGR2RGB)
        hsv_image = cv2.cvtColor(input_image,cv2.COLOR_RGB2HSV)
        output_image = zeros_like(hsv_image)
        h,s,v = cv2.split(hsv_image)
        v_copy = v.copy()
        maximum = int(np.max(v))
        cum = calculate_CDF(v,maximum,r,c)
        for i in tqdm(range(r)):
            for j in range(c):
                v_copy[i,j] = truncateHE(cum[int(v[i,j])]*maximum)

        plot_hist(input_file,input_image,v_copy)

        output_image[:,:,2] = v_copy
        output_image[:,:,0] = h
        output_image[:,:,1] = s

        output_image = cv2.cvtColor(output_image,cv2.COLOR_HSV2RGB)


    fig,axes = plt.subplots(1,2, constrained_layout=True)


    axes[0].imshow(input_image,cmap=cmap)
    axes[0].axis("on")
    axes[0].set_title("Original Image")

    im = axes[1].imshow(output_image, cmap=cmap)
    axes[1].axis("on")
    axes[1].set_title("Histogram Equalized")

    cbar = fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.35)
    plt.savefig(".."+name+"HistEq.png",bbox_inches="tight",pad=-1)

    if d==3:
        plt.imsave(".." + name+"HE.png",output_image)
    else:
        plt.imsave(".." + name+"HE.png",output_image,cmap=cmap)
```

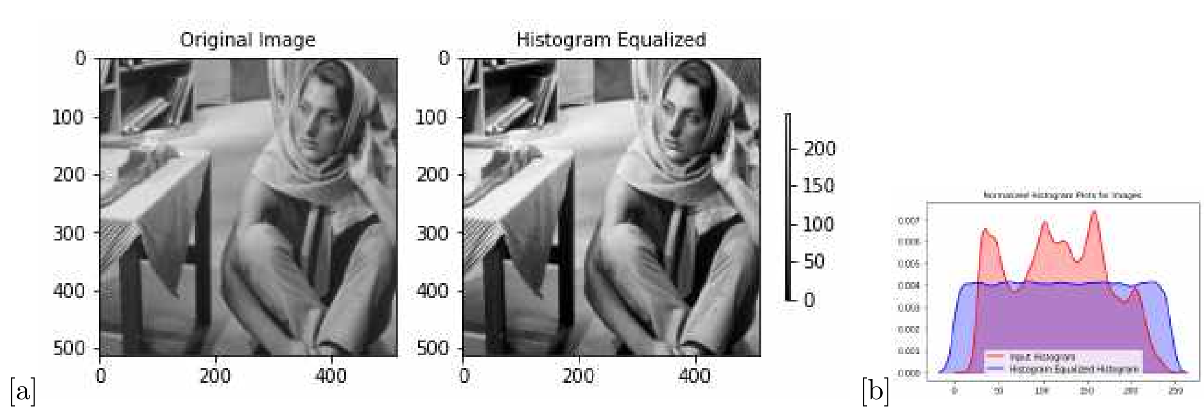
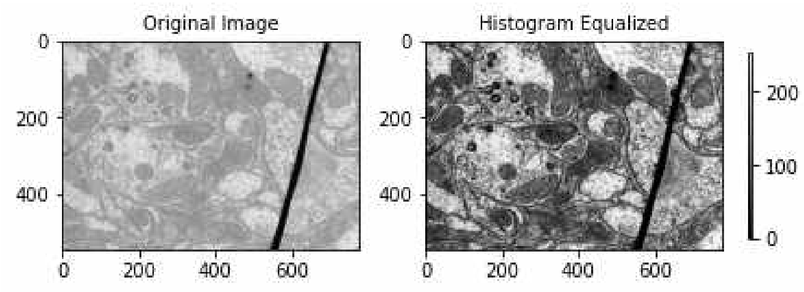


Figure 8: (a) HE for barbara.png (b) Histogram comparison for barbara.png

[c] 

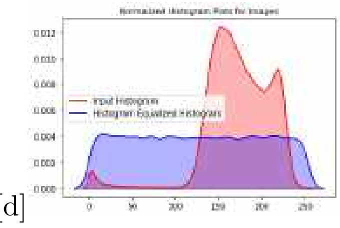
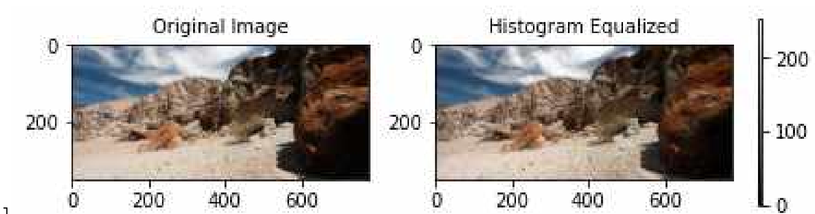
[d] 

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

[e] 

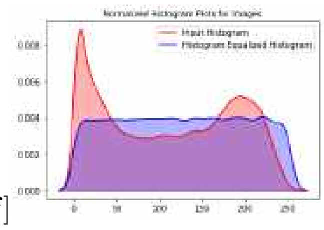
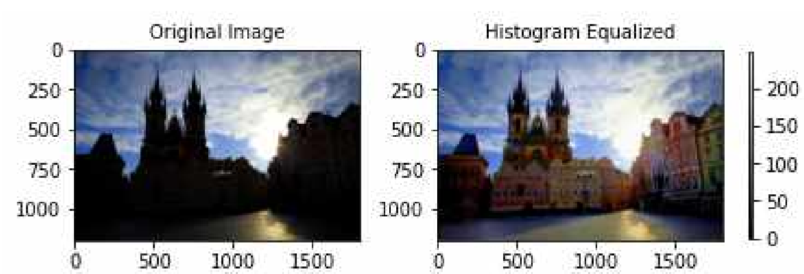
[f] 

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

[g] 

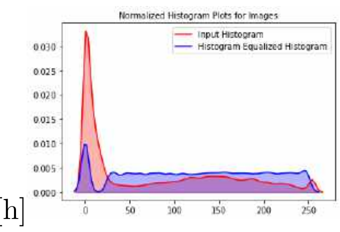
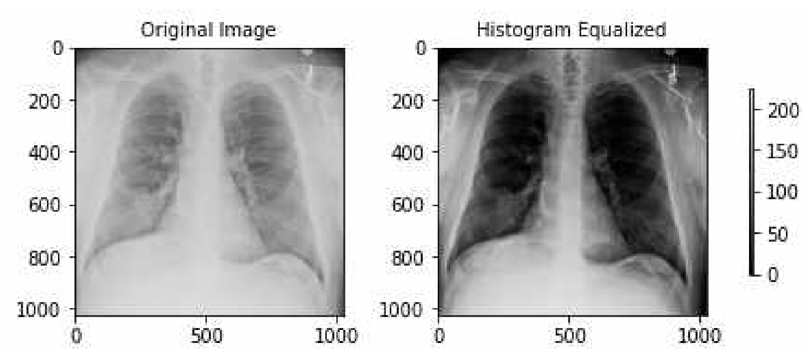
[h] 

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

[i] 

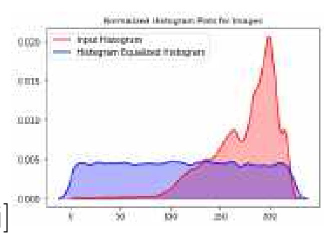
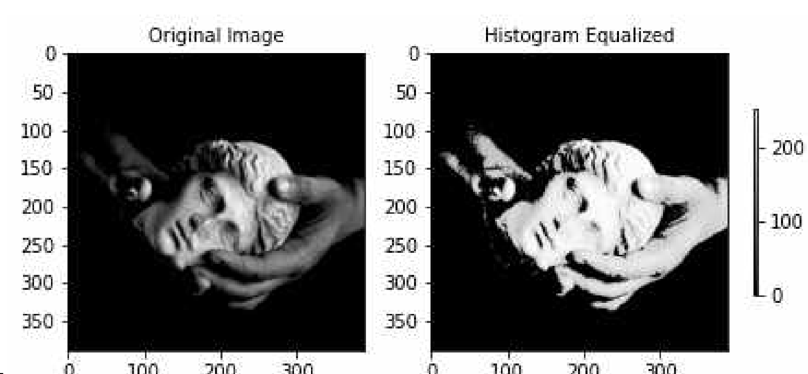
[j] 

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

[k] 

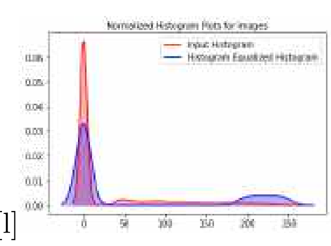
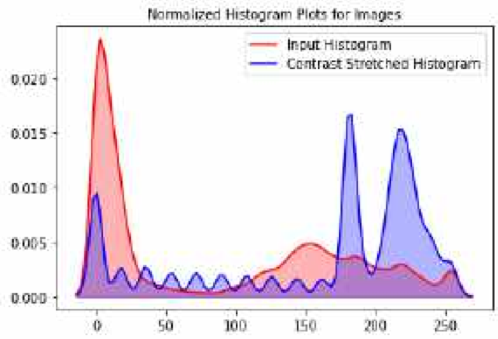
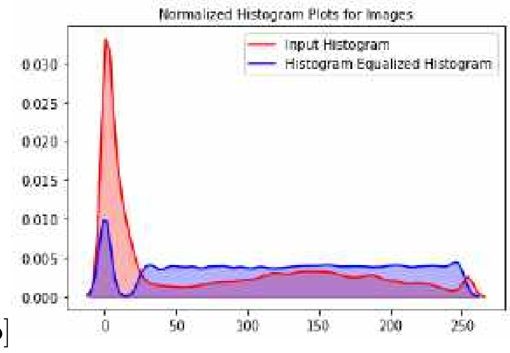
[l] 

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.



[a]



[b]

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.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import matplotlib as mpl

from tqdm import tqdm
import cv2
from seaborn import distplot
from math import floor,ceil

def plot_hist(input_file,input_image,output_image,reference_image):
    name = input_file.split(".")[2]
    plt.figure()
    plt.title("Normalized Histogram Plots for Images")
    ax = distplot(input_image,color='r',label ="Input Histogram",
         hist_kws={"alpha": 0.3, "linewidth": 1.5},bins=256,hist=False)
    ax = distplot(output_image,color="b",label ="Histogram Matched Histogram",
         hist_kws={"alpha": 0.3,"linewidth": 1.5},bins=256,hist=False)
    ax = distplot(reference_image,color='g',label ="Reference Histogram",
         hist_kws={"alpha": 0.3, "linewidth": 1.5},bins=256,hist=False)

    l1 = ax.lines[0]
    x1 = l1.get_xydata()[:,0]
    y1 = l1.get_xydata()[:,1]
    ax.fill_between(x1,y1, color="red", alpha=0.3)
    l2 = ax.lines[1]
    x2 = l2.get_xydata()[:,0]
    y2 = l2.get_xydata()[:,1]
    ax.fill_between(x2,y2, color="blue", alpha=0.3)
    l3 = ax.lines[2]
    x3 = l3.get_xydata()[:,0]
    y3 = l3.get_xydata()[:,1]
```

```python
    ax.fill_between(x3,y3, color="green", alpha=0.3)
    plt.legend()
    plt.savefig(".."+name+"HMHistogram.png",bbox_inches="tight",pad=-1)


def perform_masking(original,masking,r,c,d=3):
    """
    Masks the original image using the mask provided
    input : orig_image, mask, r(rows of orig_image), c(cols of orig_image),
            d(#channels of orig_image)
    output : masked image
    """
    orig = original.copy()
    mask = masking.copy()

    for i in range(3):
        for j in range(r):
            for k in range(c):
                orig[j,k,i] = (0 if mask[j,k,i]==0 else orig[j,k,i])

    return orig

def calculate_CDF(array,maximum,r,c):
    """
    This function is used to calculate the CDF of the 2D image.
    array : For gray scale the whole image is the input and for RGB
            each of the color slices are the inputs.
    maximum : maximum pixel intensity of the 2D image
    output : the CDF of the 2D image
    """
    freqs = np.zeros((maximum+1,1))
    probf = np.zeros((maximum+1,1))
    cum = np.zeros((maximum+1,1))

    for i in range(r):
        for j in range(c):
            freqs[int(array[i][j])]+=1

    for i,j in enumerate(freqs):
        probf[i] = freqs[i]/(r*c)

    for i,j in enumerate(probf):
        for k in range(i):
            cum[i] += probf[k]
    return cum

def myHM(reference,reference_mask,target,target_mask):

    parameters = {'axes.titlesize': 10}
    plt.rcParams.update(parameters)

    name = target.split(".")[2]

    parameters = {'axes.titlesize': 10}
    plt.rcParams.update(parameters)

    # READING THE REFERENCE IMAGE
    original_ref_image = cv2.imread(reference)
    original_ref_image = cv2.cvtColor(original_ref_image,cv2.COLOR_BGR2RGB)

    r1,c1,d1 = original_ref_image.shape
    # READING THE REFERENCE IMAGE MASK AND PRODUCING THE MASKED REFERENCE IMAGE
    original_ref_image_mask = cv2.imread(reference_mask)
    original_ref_image_mask = cv2.cvtColor(original_ref_image_mask,cv2.COLOR_BGR2RGB)
    original_ref_masked = perform_masking(original_ref_image,original_ref_image_mask,r1,c1,d1)
```

```python

    # READING THE IMAGE TO BE HISTOGRAM MATCHED
    original_target_image = cv2.imread(target)
    original_target_image = cv2.cvtColor(original_target_image,cv2.COLOR_BGR2RGB)

    r2,c2,d2 = original_target_image.shape

    # READING THE MASK OF THE IMAGE TO BE MATCHED
    original_target_image_mask = cv2.imread(target_mask)
    original_target_image_mask = cv2.cvtColor(original_target_image_mask,cv2.COLOR_BGR2RGB)
    original_target_masked = perform_masking(original_target_image,original_target_image_mask,r2,c2,d2)

    # CREATING THE HSV TRANSFORM FOR THE MASKED IMAGES AND EXTRACTING THE V-CHANNEL
    ref_hsv = cv2.cvtColor(original_ref_masked,cv2.COLOR_RGB2HSV)
    ref_v = cv2.split(ref_hsv)[2]
    target_hsv = cv2.cvtColor(original_target_masked,cv2.COLOR_RGB2HSV)
    target_v = cv2.split(target_hsv)[2]

    ref_image = ref_hsv.copy()
    target_image = target_hsv.copy()

    # CUMULATIVE DISTRIBUTION OF THE IMAGES
    cum_ref = calculate_CDF(ref_v,int(np.max(ref_v)),r1,c1)
    cum_target = calculate_CDF(target_v,int(np.max(target_v)),r2,c2)

    transform_ref=[0 for i in range(256)]
    transform_tar=[0 for i in range(256)]

    for i in range(len(cum_target)):
        transform_tar[i] = floor(255*cum_target[i])
        transform_ref[i] = floor(255*cum_ref[i])

    transform ={}

    for i in transform_tar:
        value=min(transform_ref, key=lambda x:abs(x-i))
        indx = transform_ref.index(value)
        original = transform_tar.index(i)
        transform[original]=indx

    for i in range(r1):
            for j in range(c1):
                 if(target_image[i,j,2] in transform):
                    target_image[i,j,2]=transform[target_image[i,j,2]]

    plot_hist(target,target_v,target_image[:,:,2],ref_v)

    # CONVERT BACK FROM HSV TO RGB
    target_image = cv2.cvtColor(target_image,cv2.COLOR_HSV2RGB)



    fig,axes = plt.subplots(1,3, constrained_layout=True)

    axes[0].imshow(original_ref_masked)
    axes[0].axis("on")
    axes[0].set_title("Reference Image")
    axes[1].imshow(original_target_masked)
    axes[1].axis("on")
    axes[1].set_title("Original Image")
    im = axes[2].imshow(target_image)
    axes[2].axis("on")
    axes[2].set_title("Histogram Matched")
    cbar = fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.45)
    plt.savefig(".."+name+"HistogramMatched.png",cmap="gray",bbox_inches="tight",pad=-1)
```

```
    plt.imsave(".." + name+"HM",target_image,cmap="gray")
```

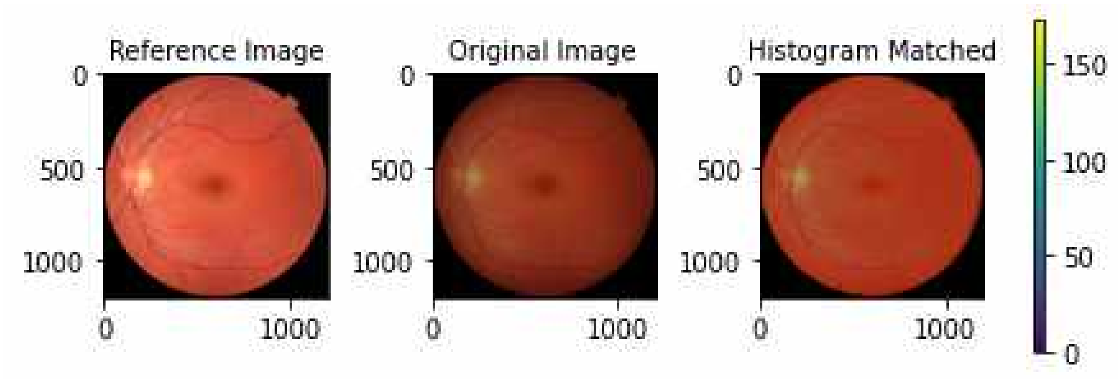


Figure 15: Output of myHM.py

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.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from numpy import zeros_like

from tqdm import tqdm
from math import floor
import cv2
from seaborn import distplot

def plot_hist(input_file,input_image,output_image,window_x,threshold):
    """
    input : input_file_path, input_image, output_image,window_size, histogram threshold
    output : saves the histograms for both the images for comparison
    dependencies : seaborn, numpy, matplotlib
```

```
    """
    name = input_file.split(".")[2]
    plt.figure()
    plt.title("Normalized Histogram Plots for Images")
    ax = distplot(input_image,color='r',label ="Input Histogram",
        hist_kws={"alpha": 0.3, "linewidth": 1.5},bins=256,hist=False)
    ax = distplot(output_image,color="b",label ="CLAHE Histogram",
        hist_kws={"alpha": 0.3,"linewidth": 1.5},bins=256,hist=False)
    l1 = ax.lines[0]
    x1 = l1.get_xydata()[:,0]
    y1 = l1.get_xydata()[:,1]
    ax.fill_between(x1,y1, color="red", alpha=0.3)
    l2 = ax.lines[1]
    x2 = l2.get_xydata()[:,0]
    y2 = l2.get_xydata()[:,1]
    ax.fill_between(x2,y2, color="blue", alpha=0.3)
    plt.legend()
    plt.savefig(".."+name+"CLAHEHistogram_"+str(window_x*2)+"_"+str(threshold)+".png",
        bbox_inches="tight",pad=-1)

def imhist(input_array):
    m, n = input_array.shape
    h = [0.0] * 256
    for i in range(m):
        for j in range(n):
            h[int(input_array[i, j])]+=1
    return np.array(h)/(m*n)

def calcCLAHEVal(input_array,th):
    r,c = input_array.shape
    H = imhist(input_array)
    C = zeros_like(H)
    for i in range(256):
        if H[i] > th:
            H[i] = th
    contrastArea = 1 - sum(H)
    height = contrastArea/256
    H = H + height
    C[0] = H[0]

    for i in range(1,256):
        C[i] = C[i-1] + (H[i])

    return C

def myCLAHE(input_file,window_x,window_y,threshold,cmap):

    parameters = {'axes.titlesize': 10}
    plt.rcParams.update(parameters)

    name = input_file.split(".")[2]
    input_image = cv2.imread(input_file)
    output_image = np.zeros_like(input_image)
    d=1
    if len(input_image.shape)<3:
        r,c = input_image.shape
    else:
        r,c,d = input_image.shape


    if d==1:
        new_image = input_image

        for i in tqdm(range(r)):
            for j in range(c):
```

```
                min_x = max(0,i-window_x)
                min_y = max(0,j-window_y)
                max_x = min(r,i+window_x)
                max_y = min(c,j+window_y)

                window_image = input_image[min_x:max_x,min_y:max_y]
                if new_image[i,j]!=0:
                    x = calcCLAHEVal(window_image,threshold)
                    output_image[i,j] = x[int(new_image[i,j])]*255

        plot_hist(input_file,input_image,output_image,window_x,threshold)

    else:
        output_hsv = np.zeros_like(input_image)
        input_image = cv2.cvtColor(input_image,cv2.COLOR_BGR2RGB)
        hsv_image = cv2.cvtColor(input_image, cv2.COLOR_RGB2HSV)
        h,s,v = cv2.split(hsv_image)

        out_v = v.copy()

        for i in tqdm(range(r)):
            for j in range(c):
                min_x = max(0,i-window_x)
                min_y = max(0,j-window_y)
                max_x = min(r,i+window_x)
                max_y = min(c,j+window_y)

                window_image = v[min_x:max_x,min_y:max_y]
                if out_v[i,j]!=0:
                    x = calcCLAHEVal(window_image,threshold)
                    out_v[i,j] = x[int(v[i,j])]*255

        plot_hist(input_file,v,out_v,window_x,threshold)

        output_hsv[:,:,0] = h
        output_hsv[:,:,1] = s
        output_hsv[:,:,2] = out_v

        output_image = cv2.cvtColor(output_hsv,cv2.COLOR_HSV2RGB)

    fig,axes = plt.subplots(1,2, constrained_layout=True)
    axes[0].imshow(input_image,cmap=cmap)
    axes[0].axis("on")
    axes[0].set_title("Original Image")
    im = axes[1].imshow(output_image,cmap=cmap)
    axes[1].axis("on")
    axes[1].set_title("CLAHE Image")
    cbar = fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.45)

    plt.savefig(".."+name+"CLAHEcombined_"+str(window_x*2)+"_"+str(threshold)+".png",
                bbox_inches="tight",pad=-1)

    plt.imsave(".." + name+"CLAHE"+str(window_x*2)+"_"+str(threshold)+".png",output_image,cmap=cmap)
```

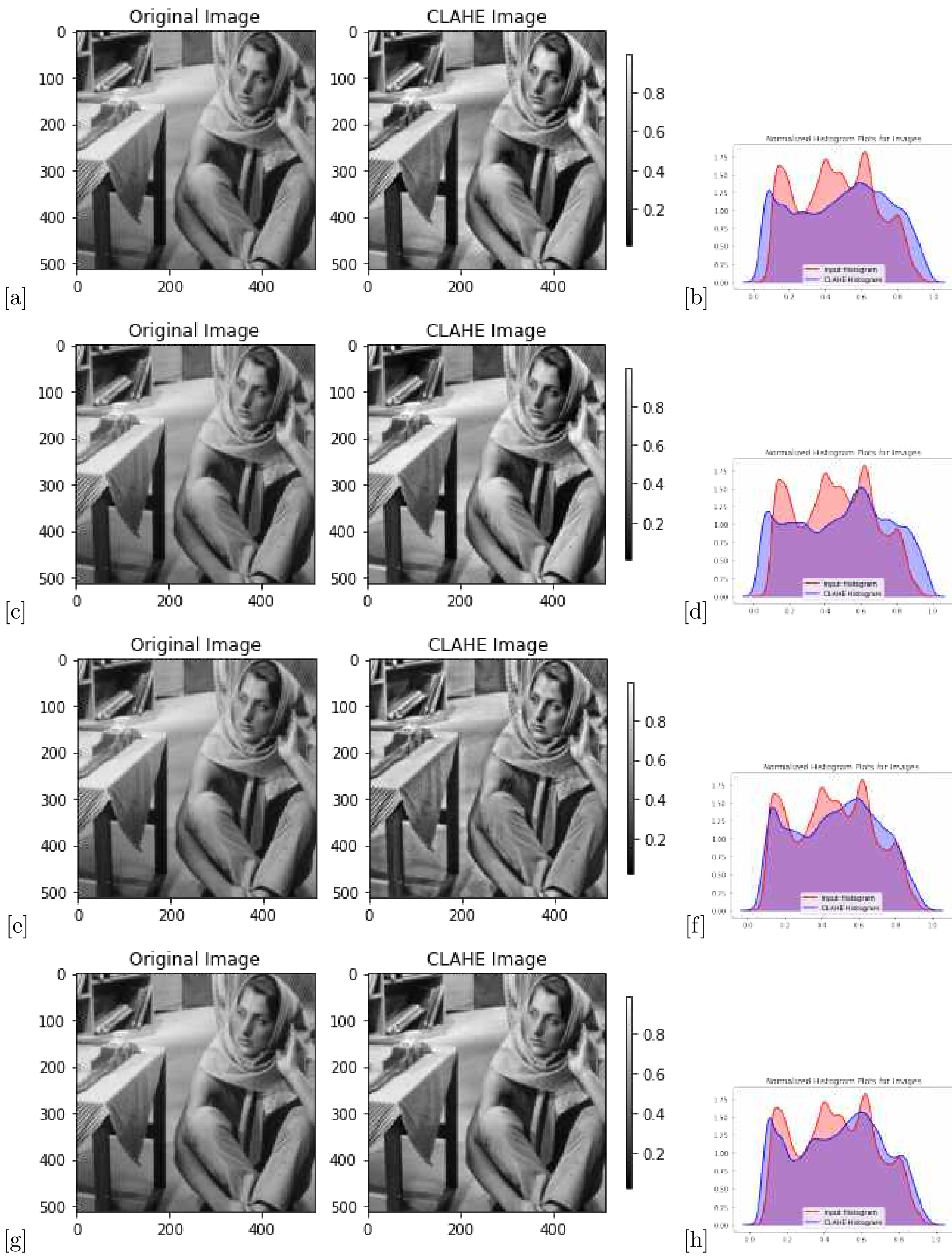


Figure 16: CLAHE operation on barbara.png with different Window Sizes and Threshold Values (a)window:128 th:0.005 (b) Histogram for (a) (c) window:256 th:0.005 (d) Histogram for (c) (e) window:32 th:0.005 (f) Histogram for (e) (g) window:128 th:0.0025 (h) Histogram for (g)

For *barbara.png* the optimum Window Size used by us is 128x128 with threshold value of 0.005

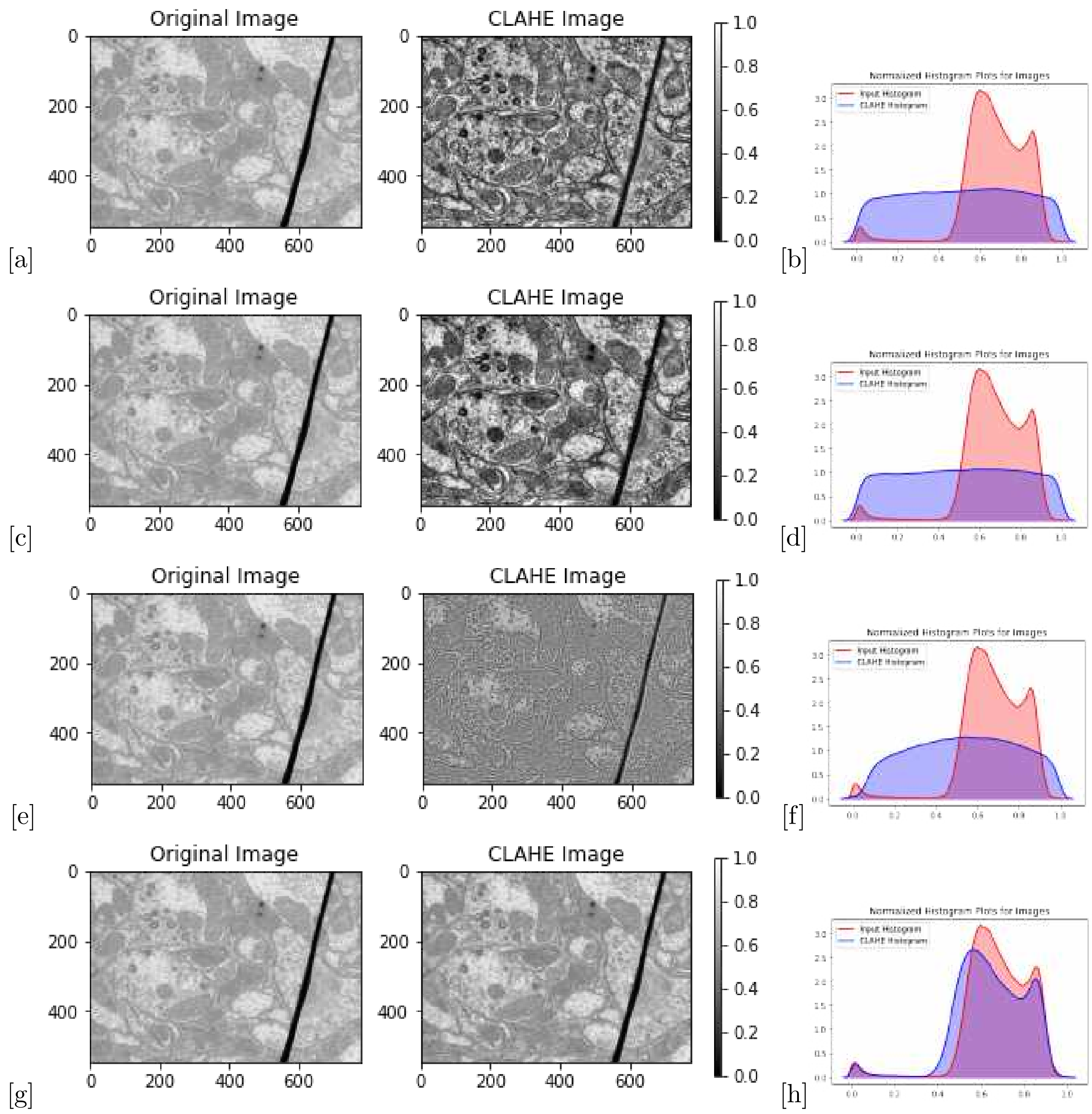


Figure 17: CLAHE operation on TEM.png with different Window Sizes and Threshold Values (a)window:64 th:0.03 (b) Histogram for (a) (c) window:128 th:0.03 (d) Histogram for (c) (e) window:8 th:0.03 (f) Histogram for (e) (g) window:64 th:0.015 (h) Histogram for (g)

For *TEM.png* the optimum Window Size used by us is 128x128 with threshold value of 0.005

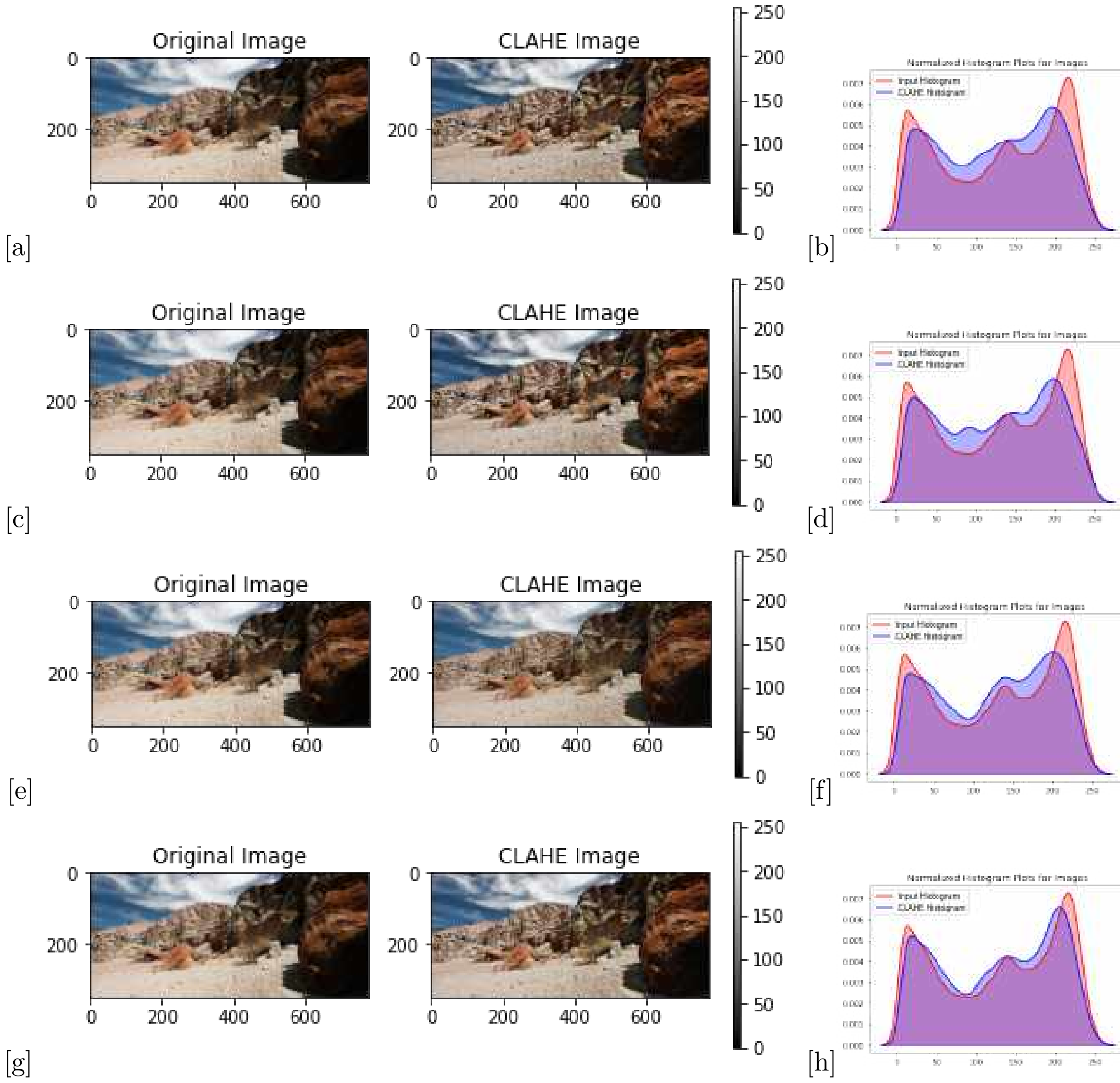


Figure 18: CLAHE operation on canyon.png with different Window Sizes and Threshold Values (a)window:32 th:0.005 (b) Histogram for (a) (c)window:64 th:0.005 (d) Histogram for (c) (e) window:8 th:0.005 (f) Histogram for (e) (g) window:32 th:0.0025 (h) Histogram for (g)

For *canyon.png* the optimum Window Size used by us is 32x32 with threshold value of 0.005

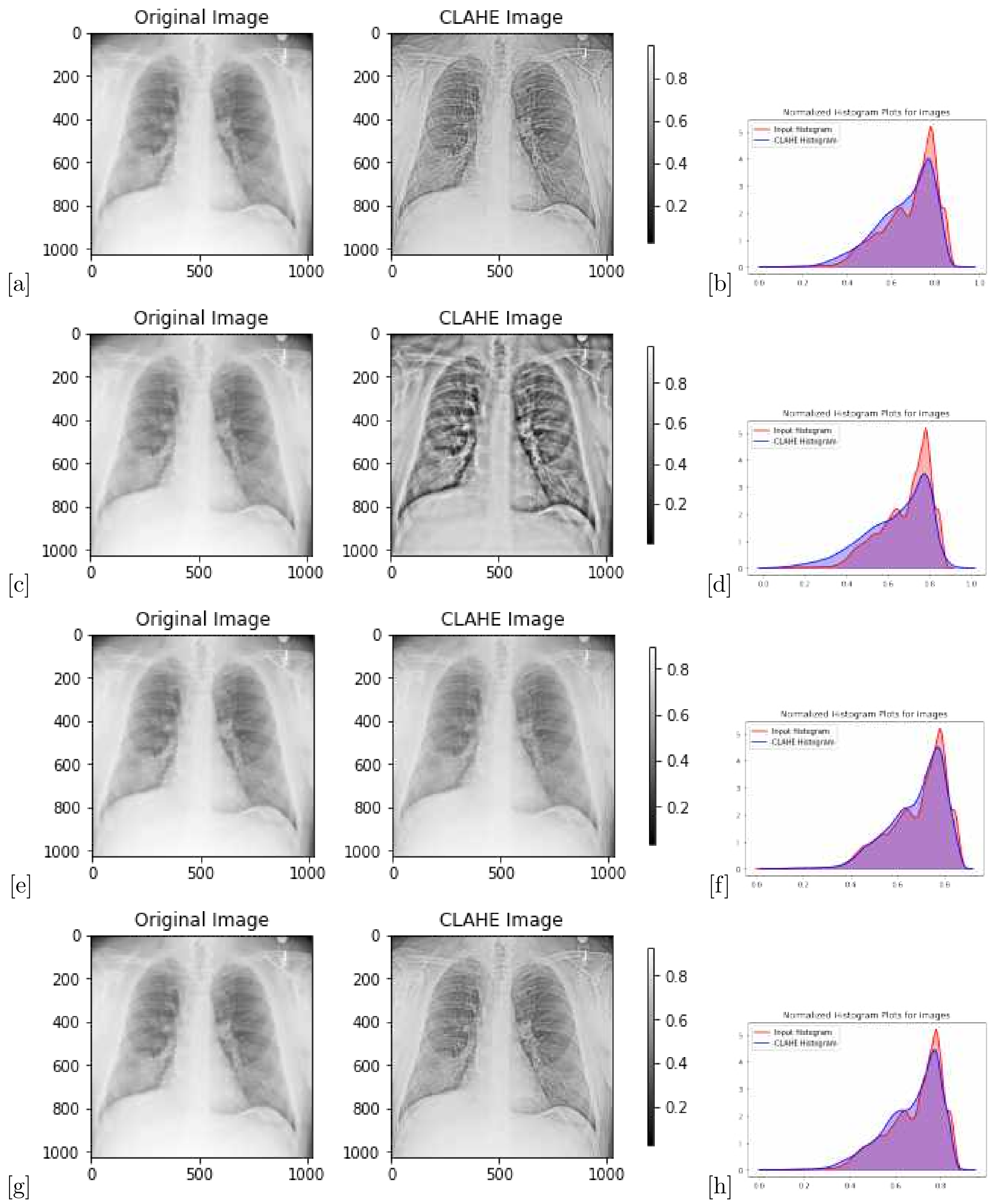


Figure 19: CLAHE operation on chestXray.png with different Window Sizes and Threshold Values (a)window:16 th:0.015 (b) Histogram for (a) (c) window:64 th:0.015 (d) Histogram for (c) (e) window:4 th:0.015 (f) Histogram for (e) (g) window:16 th:0.0075 (h) Histogram for (g)

For *chestXray.png* the optimum Window Size used by us is 16x16 with threshold value of 0.015

myMainScript.py

```python
from myHM import myHM
from myHE import myHE
from myForegroundMask import myForegroundMask
from myCLAHE import myCLAHE
from myLinearContrastStretching import myLinearContrastStretching

from time import time

image1 = "../data/barbara.png"
image2 = "../data/TEM.png"
image3 = "../data/canyon.png"
image4 = "../data/retina.png"
image5 = "../data/church.png"
```

```python
image6 = "../data/chestXray.png"
image7 = "../data/statue.png"

maskedStatue = "../data/statueForegroundMasked.png"
reference = "../data/retinaRef.png"
reference_mask = "../data/retinaRefMask.png"
target = "../data/retina.png"
target_mask = "../data/retinaMask.png"

superStart = time()
start = time()
myForegroundMask(image7)
end = time()
print("Time to run myForegroundMask.py :",end-start,"secs")

start = time()
myLinearContrastStretching(image1,[120,230],[90,245])
myLinearContrastStretching(image2,[120,230],[30,245])
myLinearContrastStretching(image3,[100,180],[60,200])
myLinearContrastStretching(image5,[10,200],[180,230])
myLinearContrastStretching(image6,[100,200],[30,180])
myLinearContrastStretching(maskedStatue,[50,200],[20,230])
end = time()
print("Time to run myLinearContrastStretching.py :",end-start,"secs")

start = time()
myHE(image1)
myHE(image2)
myHE(image3)
myHE(image5)
myHE(image6)
myHE(maskedStatue)
end = time()
print("Time to run myHE.py :",end-start,"secs")

start = time()
myHM(reference,reference_mask,target,target_mask)
end = time()
print("Time to run myHM.py :",end-start,"secs")

start = time()
myCLAHE(image1,64,64,0.005,"gray")
myCLAHE(image1,128,128,0.005,"gray")
myCLAHE(image1,16,16,0.005,"gray")
myCLAHE(image1,64,64,0.0025,"gray")
myCLAHE(image2,32,32,0.03,"gray")
myCLAHE(image2,64,64,0.03,"gray")
myCLAHE(image2,4,4,0.03,"gray")
myCLAHE(image2,32,32,0.0015,"gray")
myCLAHE(image3,16,16,0.005,"gray")
myCLAHE(image3,32,32,0.005,"gray")
myCLAHE(image3,8,8,0.005,"gray")
myCLAHE(image3,16,16,0.0025,"gray")
myCLAHE(image6,8,8,0.015,"gray")
myCLAHE(image6,32,32,0.015,"gray")
myCLAHE(image6,2,2,0.015,"gray")
myCLAHE(image6,8,8,0.0075,"gray")
myCLAHE(maskedStatue)
end = time()
print("Time to run myCLAHE.py :",end-start,"secs")

superEnd = time()
print("Time required to run codes for Q2 :",round(superEnd-superStart,2),"minutes")
```