1. Image Fundamentals

This section has three parts. first exercise is about quantization with different levels with and without histogram equalization and compare the differences.

Second exercise is about downsampling using 2 approches(averaging - remove row&col) and then unsampling using 2 approches(pixel replication - bilinear interplolation).

Third exercise is to show Elaine image with diffrenet bit planes.

First we load datasets and import libraries.

```
1 !wget --load-cookies /tmp/cookies.txt "https://docs.goo
```

```
--2022-11-25 13:19:46--  https://docs.google.com/uc?export=download&confirm=&
Resolving docs.google.com (docs.google.com)... 142.251.2.102, 142.251.2.101,
Connecting to docs.google.com (docs.google.com)|142.251.2.102|:443... connect
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-04-64-docs.googleusercontent.com/docs/securesc/ha0ro937
Warning: wildcards not supported in HTTP.
--2022-11-25 13:19:47--  https://doc-04-64-docs.googleusercontent.com/docs/se
Resolving doc-04-64-docs.googleusercontent.com (doc-04-64-docs.googleusercont
Connecting to doc-04-64-docs.googleusercontent.com (doc-04-64-docs.googleuser
HTTP request sent, awaiting response... 200 OK
Length: 786488 (768K) [image/bmp]
Saving to: ‘Barbara.bmp’

Barbara.bmp         100%[===================>] 768.05K  --.-KB/s    in 0.006s

2022-11-25 13:19:47 (121 MB/s) - ‘Barbara.bmp’ saved [786488/786488]
```

```
1 !wget --load-cookies /tmp/cookies.txt "https://docs.goo
```

```
--2022-11-25 13:19:50--  https://docs.google.com/uc?export=download&confirm=&
Resolving docs.google.com (docs.google.com)... 142.251.2.102, 142.251.2.101,
Connecting to docs.google.com (docs.google.com)|142.251.2.102|:443... connect
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-08-64-docs.googleusercontent.com/docs/securesc/ha0ro937
Warning: wildcards not supported in HTTP.
--2022-11-25 13:19:50--  https://doc-08-64-docs.googleusercontent.com/docs/se
Resolving doc-08-64-docs.googleusercontent.com (doc-08-64-docs.googleusercont
Connecting to doc-08-64-docs.googleusercontent.com (doc-08-64-docs.googleuser
HTTP request sent, awaiting response... 200 OK
Length: 263222 (257K) [image/bmp]
Saving to: ‘CameraMan.bmp’
```

```
CameraMan.bmp       100%[===================>] 257.05K  --.-KB/s    in 0.003s

2022-11-25 13:19:51 (88.1 MB/s) - ‘CameraMan.bmp’ saved [263222/263222]
```

```
!wget --load-cookies /tmp/cookies.txt "https://docs.goo
```

```
--2022-11-25 13:19:52--  https://docs.google.com/uc?export=download&confirm=&
Resolving docs.google.com (docs.google.com)... 142.251.2.102, 142.251.2.101,
Connecting to docs.google.com (docs.google.com)|142.251.2.102|:443... connect
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-04-64-docs.googleusercontent.com/docs/securesc/ha0ro937
Warning: wildcards not supported in HTTP.
--2022-11-25 13:19:52--  https://doc-04-64-docs.googleusercontent.com/docs/se
Resolving doc-04-64-docs.googleusercontent.com (doc-04-64-docs.googleusercont
Connecting to doc-04-64-docs.googleusercontent.com (doc-04-64-docs.googleuser
HTTP request sent, awaiting response... 200 OK
Length: 263224 (257K) [image/bmp]
Saving to: ‘Elaine.bmp’

Elaine.bmp          100%[===================>] 257.05K  --.-KB/s    in 0.003s

2022-11-25 13:19:52 (74.3 MB/s) - ‘Elaine.bmp’ saved [263224/263224]
```

```
!wget --load-cookies /tmp/cookies.txt "https://docs.goo
```

```
--2022-11-25 13:19:54--  https://docs.google.com/uc?export=download&confirm=&
Resolving docs.google.com (docs.google.com)... 142.251.2.102, 142.251.2.101,
Connecting to docs.google.com (docs.google.com)|142.251.2.102|:443... connect
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-0g-64-docs.googleusercontent.com/docs/securesc/ha0ro937
Warning: wildcards not supported in HTTP.
--2022-11-25 13:19:54--  https://doc-0g-64-docs.googleusercontent.com/docs/se
Resolving doc-0g-64-docs.googleusercontent.com (doc-0g-64-docs.googleusercont
Connecting to doc-0g-64-docs.googleusercontent.com (doc-0g-64-docs.googleuser
HTTP request sent, awaiting response... 200 OK
Length: 263222 (257K) [image/bmp]
Saving to: ‘Goldhill.bmp’

Goldhill.bmp        100%[===================>] 257.05K  --.-KB/s    in 0.004s

2022-11-25 13:19:54 (63.7 MB/s) - ‘Goldhill.bmp’ saved [263222/263222]
```

```
!wget --load-cookies /tmp/cookies.txt "https://docs.goo
```

```
--2022-11-25 13:19:55--  https://docs.google.com/uc?export=download&confirm=&
Resolving docs.google.com (docs.google.com)... 142.251.2.102, 142.251.2.101,
Connecting to docs.google.com (docs.google.com)|142.251.2.102|:443... connect
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-0s-64-docs.googleusercontent.com/docs/securesc/ha0ro937
Warning: wildcards not supported in HTTP.
```

```
--2022-11-25 13:19:56--  https://doc-0s-64-docs.googleusercontent.com/docs/se
Resolving doc-0s-64-docs.googleusercontent.com (doc-0s-64-docs.googleusercont
Connecting to doc-0s-64-docs.googleusercontent.com (doc-0s-64-docs.googleuser
HTTP request sent, awaiting response... 200 OK
Length: 132990 (130K) [image/jpeg]
Saving to: 'HE1.jpg'

HE1.jpg             100%[===================>] 129.87K  --.-KB/s    in 0.001s

2022-11-25 13:19:56 (120 MB/s) - 'HE1.jpg' saved [132990/132990]
```

```
!wget --load-cookies /tmp/cookies.txt "https://docs.goo
```

```
--2022-11-25 13:19:57--  https://docs.google.com/uc?export=download&confirm=&
Resolving docs.google.com (docs.google.com)... 142.251.2.102, 142.251.2.101,
Connecting to docs.google.com (docs.google.com)|142.251.2.102|:443... connect
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-08-64-docs.googleusercontent.com/docs/securesc/ha0ro937
Warning: wildcards not supported in HTTP.
--2022-11-25 13:19:57--  https://doc-08-64-docs.googleusercontent.com/docs/se
Resolving doc-08-64-docs.googleusercontent.com (doc-08-64-docs.googleusercont
Connecting to doc-08-64-docs.googleusercontent.com (doc-08-64-docs.googleuser
HTTP request sent, awaiting response... 200 OK
Length: 34705 (34K) [image/jpeg]
Saving to: 'HE2.jpg'

HE2.jpg             100%[===================>]  33.89K  --.-KB/s    in 0.001s

2022-11-25 13:19:57 (23.6 MB/s) - 'HE2.jpg' saved [34705/34705]
```

```
!wget --load-cookies /tmp/cookies.txt "https://docs.goo
```

```
--2022-11-25 13:19:58--  https://docs.google.com/uc?export=download&confirm=&
Resolving docs.google.com (docs.google.com)... 142.251.2.102, 142.251.2.101,
Connecting to docs.google.com (docs.google.com)|142.251.2.102|:443... connect
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-0s-64-docs.googleusercontent.com/docs/securesc/ha0ro937
Warning: wildcards not supported in HTTP.
--2022-11-25 13:19:59--  https://doc-0s-64-docs.googleusercontent.com/docs/se
Resolving doc-0s-64-docs.googleusercontent.com (doc-0s-64-docs.googleusercont
Connecting to doc-0s-64-docs.googleusercontent.com (doc-0s-64-docs.googleuser
HTTP request sent, awaiting response... 200 OK
Length: 104598 (102K) [image/jpeg]
Saving to: 'HE3.jpg'

HE3.jpg             100%[===================>] 102.15K  --.-KB/s    in 0.001s

2022-11-25 13:19:59 (104 MB/s) - 'HE3.jpg' saved [104598/104598]
```

```
!wget --load-cookies /tmp/cookies.txt "https://docs.goo
```

```
--2022-11-25 13:20:00--  https://docs.google.com/uc?export=download&confirm=&
Resolving docs.google.com (docs.google.com)... 142.251.2.102, 142.251.2.101,
Connecting to docs.google.com (docs.google.com)|142.251.2.102|:443... connect
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-0c-64-docs.googleusercontent.com/docs/securesc/ha0ro937
Warning: wildcards not supported in HTTP.
--2022-11-25 13:20:00--  https://doc-0c-64-docs.googleusercontent.com/docs/se
Resolving doc-0c-64-docs.googleusercontent.com (doc-0c-64-docs.googleusercont
Connecting to doc-0c-64-docs.googleusercontent.com (doc-0c-64-docs.googleuser
HTTP request sent, awaiting response... 200 OK
Length: 154448 (151K) [image/jpeg]
Saving to: ‘HE4.jpg’

HE4.jpg             100%[===================>] 150.83K  --.-KB/s    in 0.001s

2022-11-25 13:20:00 (107 MB/s) - ‘HE4.jpg’ saved [154448/154448]
```

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import sklearn
4 import cv2
5 from math import *
```

1.1. Quantization & Interpolation

1.1.1.

First Elaine image is quantized in 4,8,16,32,64,128 levels. the algorithm for quantization is to first get the final gray levels(n_final_gray_levels). consider 2^k = (256/n_final_gray_levels) so k is the amount of bits that we have to shift image to right to have n_final_gray_levels of quantization. by img / (256/n_final_gray_levels) we actually shift img k bits to right. we have n_final_gray_levels of quantization level.now we have to map this gray levels to 0-255. to do this we multiply output image by np.floor(255/(n_final_gray_levels -1)).

For example to have 32 gray levels(5 quantization bits). by img/(256/32), we shift image 3 bits to the right to just have 5 quantization bits and therefore have 32 grayscales. now every gray level values are between 0-31, to map these numbers to 0-255, we have to multiple image by 255/32 which can be calculated by np.floor(255/(n_final_gray_levels -1))

```
def Quantize(img,n_final_gray_levels):
  output= np.floor(img / (256/n_final_gray_levels))
  output= output * np.floor(255/(n_final_gray_levels -1
  return output
```

```
img=cv2.imread('Elaine.bmp',cv2.IMREAD_GRAYSCALE)
plt.imshow(img,cmap='gray')
plt.axis('off')
plt.show()
```

```
#Quantize with 4 8 16 32 64 128 gray levels:
img_4=Quantize(img,4)
img_8=Quantize(img,8)
img_16=Quantize(img,16)
img_32=Quantize(img,32)
img_64=Quantize(img,64)
img_128=Quantize(img,128)
```

plot grid is a function that takes images as input and a list of quntization levels. it first shows each image and then calls np.histogram to compute histigram of images and plot histograms as well. plot grid uses quntization levels for number of bins in calculating histograms.

```
def plot_grid(*args, figsize=10,title=None, fontsize=12
    figsize = (3*figsize, figsize)
    #figsize = (figsize, figsize)
    images =list(args)
    fig=plt.figure(figsize=figsize)
    for i in range(1, len(images)+1):

        fig.add_subplot(2, len(images), i)
```

```
        if title is not None:
            plt.title(title[i-1], fontsize=fontsize)
        plt.imshow(images[i-1],cmap='gray')
        plt.axis('off')

        ## plot histograms:
        if quan_levels != None:
          fig.add_subplot(2, len(images), i+len(images)
          img_hist=np.histogram(images[i-1].ravel(),qua
          plt.bar(np.linspace(0,255,quan_levels[i-1]),i
```

Results of quantization without histogram equalization on image

By using more bits as quantization bits it is shown that we can see a broader range of gray values in image and the image has more quality. cause by using more gray levels, contrast(variance) as well as quality of image is going to increase.

In histograms, number of bins is equal to number of quantization levels.

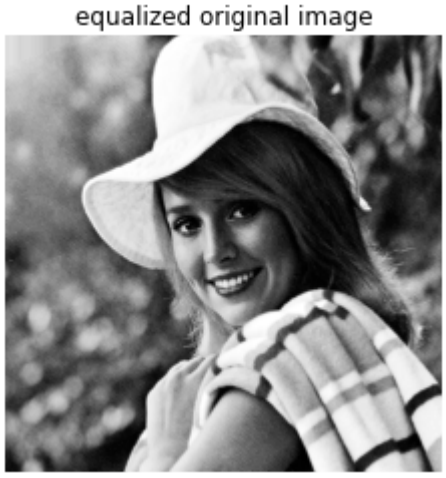
```
titles=['img_4 Quantized','img_8 Quantized','img_16 Qua
plot_grid(img_4,img_8,img_16,img_32,img_64,img_128,titl
```

MSE Function

Mean squared error computes difference between two images by averaging of squared difference between pixels of two images.

```
def mse(img1,img2):
  pic_1=img1.copy().ravel()
  pic_2=img2.copy().ravel()
  sqr_diff=(pic_1 - pic_2)**2
  size=pic_1.shape[0]
  err = np.sum(sqr_diff) / size
  return err

```

```
img_equ = cv2.equalizeHist(img)
plt.imshow(img_equ,cmap='gray')
plt.title('equalized original image', fontsize=12)
plt.axis('off')
plt.show()
```



Results of quantization with histogram equalization on image

Same as quantization without histogram equalization by using more bits as quantization bits it is shown that we can see a broader range of gray values in image

and the image has more quality. cause by using more gray levels, contrast(variance) as well as quality of image is going to increase.

In histograms, number of bins is equal to number of quantization levels.

The diffrence between with or without equalization is that, cause equlization tries to change the intensity values in a way that the corresponding histigram has the uniform distrebution when we apply quantization on equalized image, the corresponding histograms have same horizontal axis as histograms before equalization but the y-axis is different. y axis in histograms after equalization are really close to uniform function(because of equalization) but y axis in histograms without equalization areen't uniform.

```
img_4_equ=Quantize(img_equ,4)
img_8_equ=Quantize(img_equ,8)
img_16_equ=Quantize(img_equ,16)
img_32_equ=Quantize(img_equ,32)
img_64_equ=Quantize(img_equ,64)
img_128_equ=Quantize(img_equ,128)
plot_grid(img_4_equ,img_8_equ,img_16_equ,img_32_equ,img
```

Results of mse on quantized image with and without histogram equalization and original image

---

comparision between with and without equalization:

Mse between original image and quantized without equalization for every quantization level is always less than mse between original image and quantized with equalization for every quantization level. the reason for this is that equalization

changes original intensity values to make histogram distrobution close to uniform distrobution. changing original intensity causes much more mse with original image.

comparision between different quantization levels in both cases(with or without equalization):

To compare mse between diffrenet quatized levels without quantization we can see that the least error belongs to quantization with 128 quantization levels.in other word mse is reducing by adding quantization levels. the explanation for this is prettu easy, by adding more quantization levels, we are using more bits for each pixel and ther for number og gray levels increses which causes boost of contrast, increses quality of image and nakes it more similar to the original image which means less mse.this comparison is true for both cases(with or without equalization)

```
#without_histeq
without_histeq_4 = mse(img,img_4)
without_histeq_8 = mse(img,img_8)
without_histeq_16 = mse(img,img_16)
without_histeq_32= mse(img,img_32)
without_histeq_64= mse(img,img_64)
without_histeq_128= mse(img,img_128)

#with_histeq
with_histeq_4 = mse(img,img_4_equ)
with_histeq_8 = mse(img,img_8_equ)
with_histeq_16 = mse(img,img_16_equ)
with_histeq_32= mse(img,img_32_equ)
with_histeq_64= mse(img,img_64_equ)
with_histeq_128= mse(img,img_128_equ)

print('mse:     without_histeq_4 : ',without_histeq_4,'
print('mse:     without_histeq_8 : ',without_histeq_8,'
print('mse:     without_histeq_16 : ',without_histeq_16
print('mse:     without_histeq_32: ',without_histeq_32,
print('mse:     without_histeq_64: ',without_histeq_64,
print('mse:     without_histeq_128 : ',without_histeq_1
```

```
mse:     without_histeq_4 :  856.5056190490723       with_histeq_4 :  2958.07
mse:     without_histeq_8 :  148.96257781982422       with_histeq_8 :  1573.6
mse:     without_histeq_16 :  32.774803161621094       with_histeq_16 :  1185
```

```
mse:     without_histeq_32:  17.381858825683594       with_histeq_32 :  969.5
mse:     without_histeq_64:  3.4688339233398438       with_histeq_64:  923.77
mse:     without_histeq_128 :  0.5092544555664062       with_histeq_128 :  90
```

Check mse value from sklearn.metrics to check correctness of our mse function. the mse between img and img_4_equ is the same as our result wich means our mse function works just fine.

```
# check with sklearn for mse for with_histeq_4 --> mse
from sklearn.metrics import mean_squared_error
mean_squared_error(img,img_4_equ)
```

```
2958.073097229004
```

link text

1.1.2.

In this exercise we first downsample goldhill image using averaging and removing row&col. them we upsample the downsampled images and caclulate mse for results.

First, goldhill omage is loaded.

```
goldhill_img=cv2.imread('Goldhill.bmp',cv2.IMREAD_GRAYS
plt.imshow(goldhill_img,cmap='gray')
plt.axis('off')
plt.show()
```

```
goldhill_img.shape
```

```
(512, 512)
```

avg_downsample Function

Averaging downsampling is actually the same as appling a box filter to picture. avg_downsample applies box filter to image(averaging).

avg_downsample gets size of window used for averaging or kernel(filter) size k. cause averging sums all pixel values together and then divides the sum out put by size of pixels in window(k^2), we consider a kernel like box filter by np.ones((k,k)) / (k**2).

consider size of iamge n and size of kernel k, the output size will be floor(n/k). by looping over rows and cols of out put image, we calculate it's value by appling ;ernel to corresponding patch for that pixel.

```
def avg_downsample(img,k):
  n=img.shape[0]
  kernel=np.ones((k,k)) / (k**2)
  out_size=floor(img.shape[0]/k)
  out = np.zeros((out_size,out_size))
  for i  in range(0,out_size):
    for j in range(0,out_size):
      patch = img[i*k:(i+1)*k, j*k:(j+1)*k]
      out[i,j] = np.sum(patch * kernel)
  return out
```

remove_row_col_downsampling Function

Gets downsampling factor as input and skip rows and columns by step of factor.

```
def remove_row_col_downsampling(img,factor):
  return img[::factor,::factor]
```

```
avg_goldhill_img = avg_downsample(goldhill_img,2)
remove_row_col_goldhill_img = remove_row_col_downsampli
```

```
titles=['avg_goldhill_img','remove_row_col_goldhill_img
plot_grid(avg_goldhill_img,remove_row_col_goldhill_img,
```

avg_goldhill_img

remove_row_col_goldhill_img

unsample_px_replication Function

This function is the implementation of interpolation based on pixel replication.in Pixel replication or (Nearest neighbor interpolation), each pixel gets intensity value of it's nearest neighbor.

Each pixel is replicated in this method factor(unsampling factor) times times row wise and column wise and you got a zoomed image.

```
def unsample_px_replication(img, factor):
    out = np.repeat(img, factor, axis=0)
    out = np.repeat(out, factor, axis=1)
    return out
```

bilinear_interpolation_px Function

This function is the implementation of interpolation based on pixel replication.in Pixel replication or (Nearest neighbor interpolation), each pixel gets intensity value of it's nearest neighbor.

for each pixel, 4 neigbours are calculated, then the weights are calculated using page 86 in slides.

```
def bilinear_interpolation_px(img,row_pos,col_pos):

  row_before_pos = floor(row_pos)
  row_after_pos = min(row_before_pos + 1 , img.shape[0]
  col_before_pos = floor(col_pos)
  col_after_pos = min(col_before_pos + 1 , img.shape[1]

  weights = []
  a = img[row_before_pos , col_before_pos]
  b = img[row_before_pos , col_after_pos]
  d = img[row_after_pos , col_before_pos]
```

```
  c = img[row_after_pos ,col_after_pos ]
# calculate weights
  wa = (row_after_pos - row_pos) * (col_after_pos - col
  wb = (row_after_pos - row_pos) * (col_pos - col_befor
  wc = (row_pos - row_before_pos) * (col_pos - col_befo
  wd = (row_pos - row_before_pos) * (col_after_pos - co

  # soert weights
  weights.append(wa)
  weights.append(wb)
  weights.append(wc)
  weights.append(wd)
  weights.sort(reverse=True)


  return weights[0]*a + weights[1]*b + weights[2]*c + w
```

```
def bilinear_interpolation_img(img,factor):
  out = np.zeros((img.shape[0]*factor ,img.shape[1]*fac
  for i in range(0,out.shape[0]):
    for j in range(0,out.shape[1]):
      out[i,j] = bilinear_interpolation_px(img,i/factor
  return out
```

```
bilinear_unsample_avg_img = bilinear_interpolation_img(
bilinear_unsample_remove_row_col_img = bilinear_interpo

px_replication_unsample_avg_img = unsample_px_replicati
px_replication_unsample_remove_row_col_img = unsample_p
```

```
plt.imshow(bilinear_unsample_avg_img,cmap='gray')
plt.axis('off')
plt.show()
```

Results after upsampling

By comparing mse in these 4 picyures, it is obvious that the remove orw & col downsampled image with pixel replication has the least mse. and it,s picture is the most similar one to the original picture.

In all case, pictures with remove row&col downsampling has less mse than averaging.

In all case, pictures with pixel replication(nearest neighbour) upsampling less mse than bilinear interpolation.

remove row&col has better results for downsampling and pixel replication has better results for upsampling.

```
titles=['bilinear_unsample_avg_img','px_replication_uns
plot_grid(bilinear_unsample_avg_img,px_replication_unsa
```


```
mse_avg_replication = mse(goldhill_img ,px_replication_
mse_avg_bilinear = mse(goldhill_img,bilinear_unsample_a
mse_remove_row_col_replication = mse(goldhill_img ,px_r
mse_remove_row_col_bilinear = mse(goldhill_img ,bilinea

print('mse :  mse_avg_replication ', mse_avg_replicatio
print('mse :  mse_remove_row_col_replication ', mse_rem
```

```
mse :  mse_avg_replication  65.97711181640625           mse_avg_bilinear:
mse :  mse_remove_row_col_replication  23.089599609375   mse_remove_row_col_b
```

1.1.3.

In this exercise we test another piece wise linear transformayion function, bit plane.

The bit_plane function is the implementation of bit plane. each pixel in pictures has an 8-bits value. in bit_plane, each picture is shown using specific bits in input. means that for every intensity value the specific bits are 1 and others are 0.

bit_pane function gets a binary number and then it applies logical and for every pixel value and binary number.

For example for showing pixels value only using MSB the binary number as input for bit_plan efunction should be 0b10000000.

```
barbara_img=cv2.imread('Barbara.bmp',cv2.IMREAD_GRAYSCA
plt.imshow(barbara_img,cmap='gray')
plt.axis('off')
plt.show()
```

```
def bit_plane(img,bit_plane):
  out = np.zeros((img.shape[0],img.shape[1]))
  for i in range(0,img.shape[0]):
    for j in range(0,img.shape[1]):
      out[i,j] = img[i,j] & bit_plane
  return out
```

```
img_bit_plane_76543= bit_plane(barbara_img , 0b1111100
img_bit_plane_7654= bit_plane(barbara_img , 0b11110000)
img_bit_plane_765= bit_plane(barbara_img , 0b11100000)
img_bit_plane_76= bit_plane(barbara_img , 0b11000000)
img_bit_plane_7= bit_plane(barbara_img , 0b10000000)
```

Results of showing image with diffrerent bit planes - part 1

By using bit 7(MSB) only general aspects of image are showm wich means MSB contains general information. so its not possible to see the details inte background.

for example in roiginal image in the floor between chair and table, some places are darker than the others, it is sort of like a shadow. but in img_bit_plane_7 that area is just white, confirmaing that more valuable bits cintains general information and not edges,changes and details.

By adding bits number 6,5,4 we can see that by adding each of this bits to bit plan, the background isgetting more complete. in img_bit_plane_7 background is completely flat with no details, by adding bit 6, in img_bit_plane_76 background is not flat anymore and it gained some details.

By adding bit 5 in img_bit_plane_765 the background has more details in backround but it still has perceptible false contouring.this effect can be reduced by adding the 4 bit plane.

```
titles=['img_bit_plane_76543','img_bit_plane_7654','img
plot_grid(img_bit_plane_76543,img_bit_plane_7654,img_bi
```

Results of showing image with diffrerent bit planes - part 2

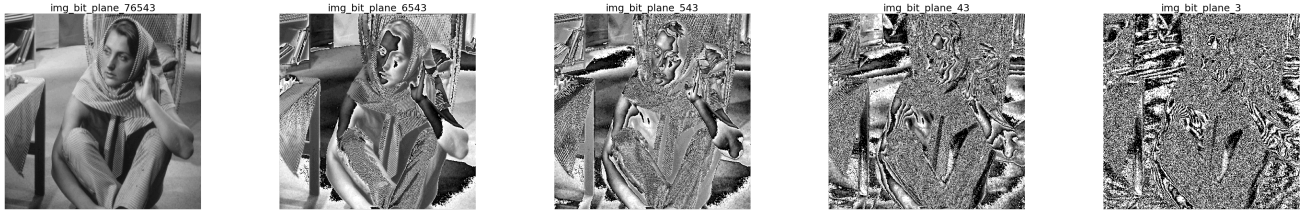
By using bits 7,6,5,4,3 once in a time, by img_bit_plane_7 it is determined that msb contains generals.

Looking through img_bit_plane_6, img_bit_plane_5,img_bit_plane_4,img_bit_plane_3 shows that lower bits contains places with high frequency, full of changes noises.

img_bit_plane_3 only contains high frequencies and edges and noises and there is no sign of basic and general parts of the image. the original image is not specified any more.

In conclusion more significant bits are for general parts (more important for recognising the image) and less significant bits for edges, changes, details.

```
img_bit_plane_76543= bit_plane(barbara_img , 0b11111000
img_bit_plane_6543= bit_plane(barbara_img , 0b01111000)
img_bit_plane_543= bit_plane(barbara_img , 0b00111000)
img_bit_plane_43= bit_plane(barbara_img , 0b00011000)
img_bit_plane_3= bit_plane(barbara_img , 0b00001000)

titles=['img_bit_plane_76543','img_bit_plane_6543','img
plot_grid(img_bit_plane_76543,img_bit_plane_6543,img_bi
```




```
img_bit_plane_76543= bit_plane(barbara_img , 0b11111000
img_bit_plane_7= bit_plane(barbara_img , 0b10000000)
img_bit_plane_6= bit_plane(barbara_img , 0b01000000)
img_bit_plane_5= bit_plane(barbara_img , 0b00100000)
img_bit_plane_4= bit_plane(barbara_img , 0b00010000)
img_bit_plane_3= bit_plane(barbara_img , 0b00001000)

titles=['img_bit_plane_76543','img_bit_plane_7','img_bi
plot_grid(img_bit_plane_76543,img_bit_plane_7,img_bit_p
```

```
out  = cv2.pyrDown(goldhill_img)
```

```
1 out.shape
```

```
(256, 256)
```

```
1 out_1 =  cv2.resize(avg_goldhill_img, (512, 512), inter
```

```
1  mse(goldhill_img ,out_1 )
```

```
61.81769633293152
```

```
1 plt.imshow(out,cmap='gray')
2 plt.axis('off')
3 plt.show()
```

2. Contrast Adjustment

2.1. Histogram Equalization

2.1.1.

First load the CameraMan image.

In this part, calculate_histogram function, calculates histagram of a gray scale picture.

calculate_histogram loops all over pixels and assign one bin for each gray level. for every gray level value the number of accurances of this value increses by one(increse value of the corresponding bin).

The stem plot of camera man histogram is plotted below. it has 2 peaks, one in low gray values (dark) with maximum 6000 value for the bin which corresponds to the black coat and hair of camaera man and dark part of his camera. the other peak is around 180 which correspond mainly to the light background.

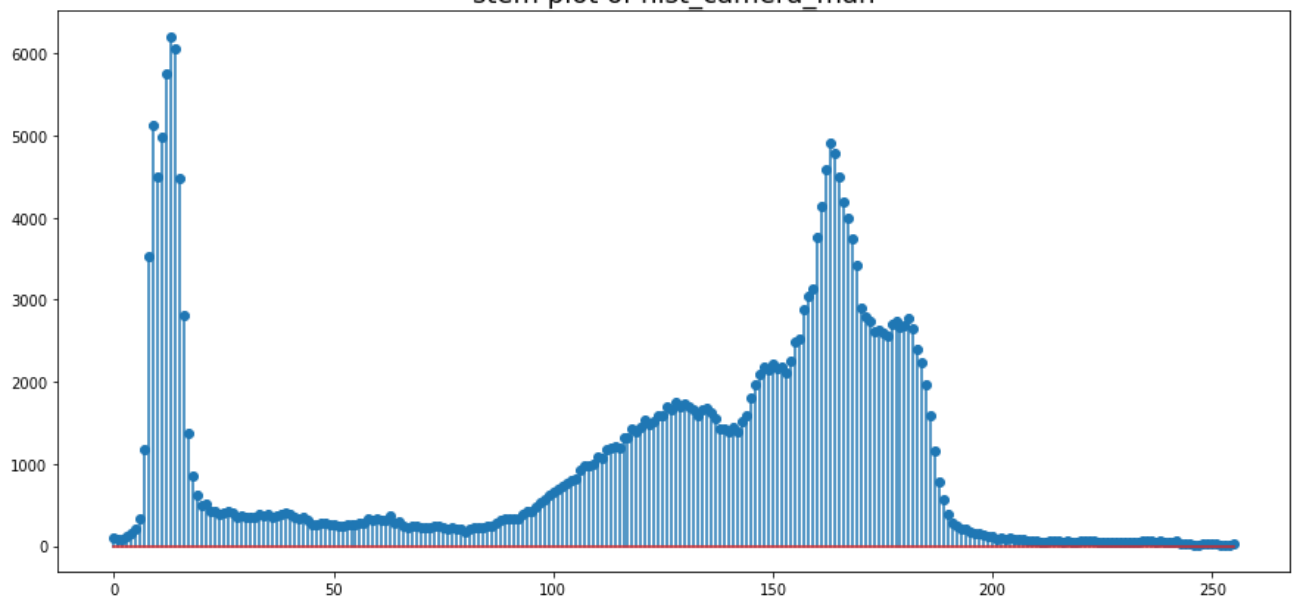
```
camera_man_img=cv2.imread('CameraMan.bmp',cv2.IMREAD_GF
plt.imshow(camera_man_img,cmap='gray')
plt.axis('off')
plt.show()
```

```
def calculate_histogram(img):
  hist = np.zeros((256))   # kepps counts of each gl (i
  for i in range(0,img.shape[0]):
    for j in range(0,img.shape[0]):
      hist[img[i,j]]+=1
  return hist
```

```
hist_camera_man = calculate_histogram(camera_man_img)
```

```
plt.figure(figsize=(15,7))
plt.title('stem plot of hist_camera_man',fontsize=18)
plt.stem(np.linspace(0,255,256),hist_camera_man)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: UserWarning:
  This is separate from the ipykernel package so we can avoid doing imports u
```



2.1.1.1.

Using division, we can reduce the brightness of a picture.change_intensity function, multiplies change_scale by each gray value. change_scale less than 1 is same as division.

```
def change_intensity(img,change_scale):
  out = img.copy()
  for i in range(0,out.shape[0]):
    for j in range(0,out.shape[1]):
      out[i,j] = max(change_scale*img[i,j],0)
      out[i,j] = min(change_scale*img[i,j],255)

  return out
```

```
D = change_intensity(camera_man_img , 1/3)
D[0,0]=240
plt.figure(figsize=(7,7))
plt.imshow(D,cmap='gray')
plt.axis('off')
plt.show()
```

2.1.1.2.

By plotting darkened image D, we can see that the quality of image for human eyes is reduced as well as contrast.

By plotting histopgram darkened image D and original image, we can see that all bins in D histograms are before 100(dark region) but original image histogram has bins in much larger range from 0 to 200(dark to light).

D histogram is dense in small range of values and All bins in D histogram are really close together (small range) while original image histogram has more dispersed bins in much larger range which explains low contrast in D(smaller range of gray values) and more contrast in riginal image(wider range of gray values)

```
1 hist_darker_camera_man = calculate_histogram(D)
```

```
1 hist_darker_camera_man.shape
```

```
(256,)
```

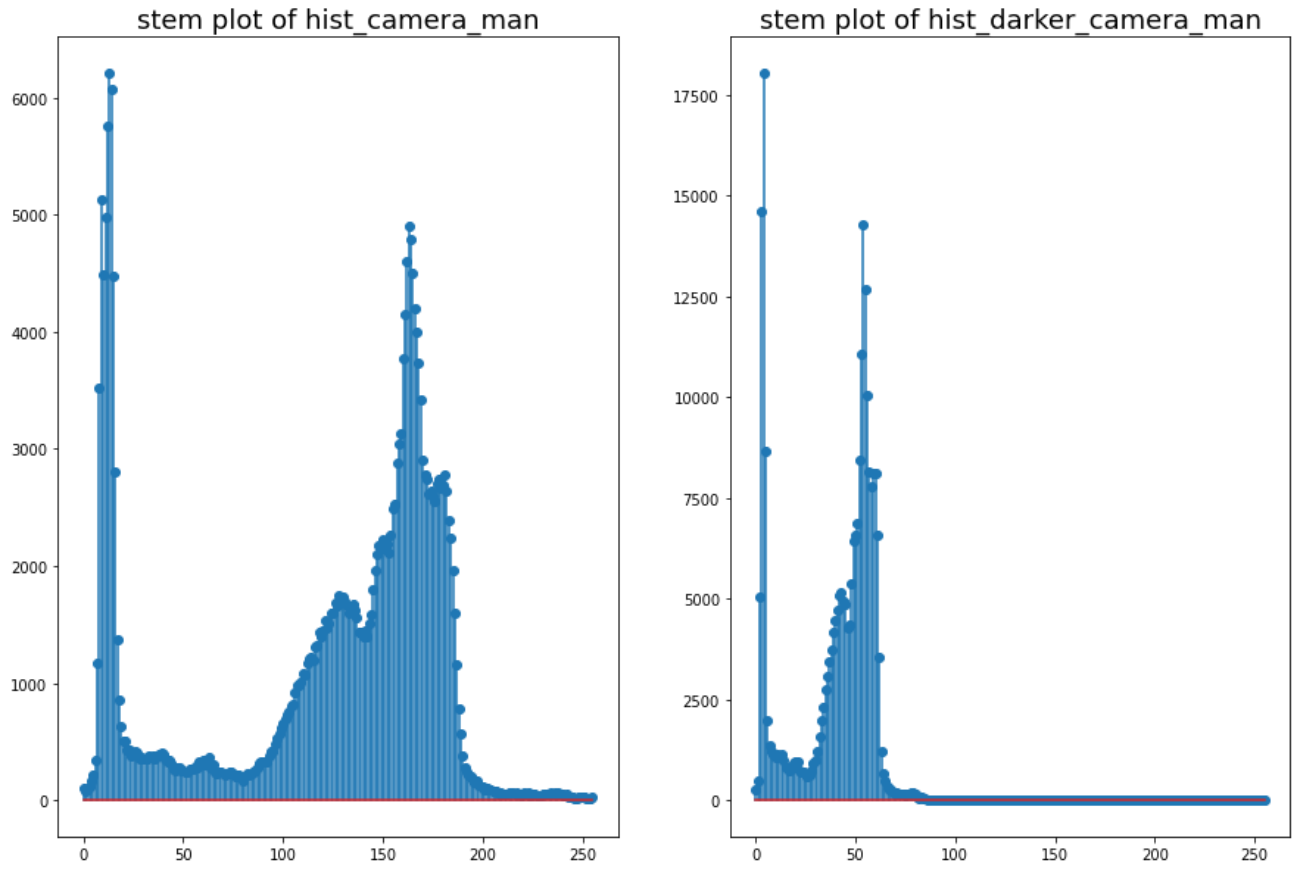
```
1 hist_darker_camera_man.shape[0]
2 #camera_man_img.shape
```

```
256
```

```
1 fig = plt.figure(figsize=(15, 10))
```

```
fig.add_subplot(1, 2, 1)
plt.title('stem plot of hist_camera_man',fontsize=18)
plt.stem(np.linspace(0,255,256),hist_camera_man)

fig.add_subplot(1, 2, 2)
plt.title('stem plot of hist_darker_camera_man',fontsiz
plt.stem(np.linspace(0,255,256),hist_darker_camera_man)

plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: UserWarning:
  after removing the cwd from sys.path.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
```



2.1.1.3.

Extract

Histogram Equalization is a technique that changes histogram distribution to uniform distribution. histogram with uniform distribution has bins in almost all gray level values and length of all bins is almost the same. meaning that this picture has almost all gray levels.

Histogram Equalization on a picture increases gray levels in a picture to close the distribution to uniform distribution thus the contrast and quality of image increases. it is a technique for boosting contrast in low contrast images.

Implementation

hist_equalization function implements histogram equalization. first it calculates histogram of an image by using calculate_histogram function from previous part then sums up all bins in histogram to get the total number of gray levels in image(mn). Then the normalized hisftogram or probability density function(pdf) is calculated by deviding histogram to mn(total number of gray levels in image).

Then the commulative density function for histogram is calculated by using np.cumsum on pdf. by multiplying max intensity value (for mapping bins) in picture by cdf we have the equalized version of histogram.

Results

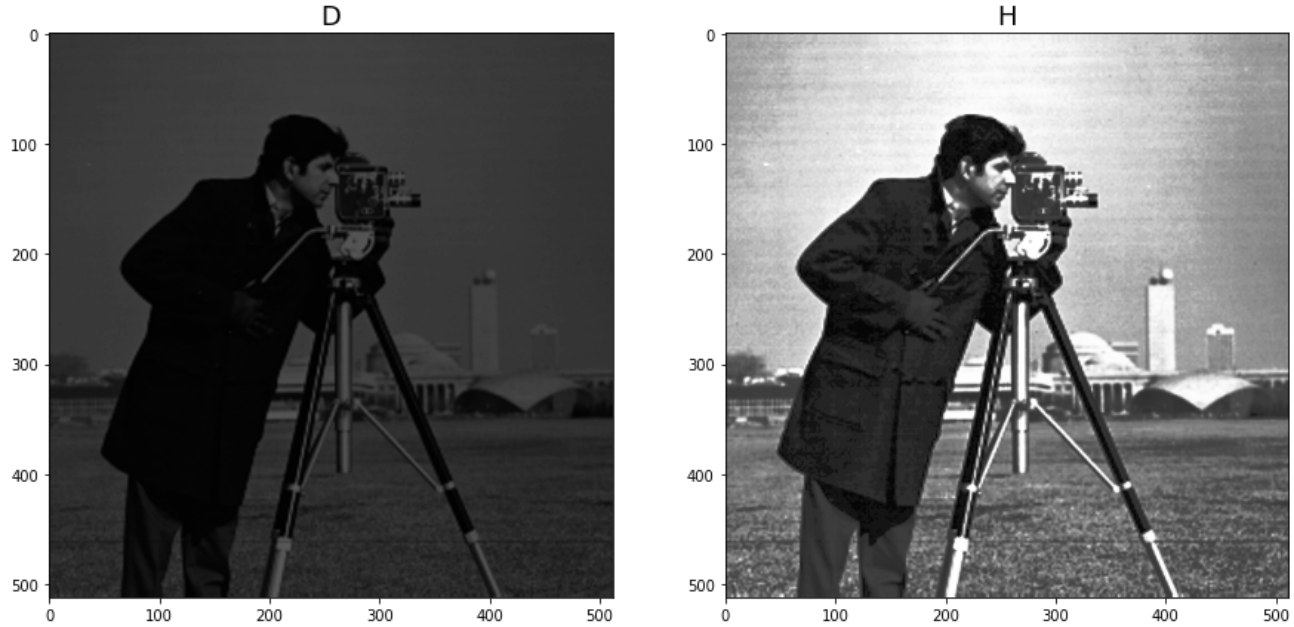
H is the equalized version of D by performing hist_equalization(D). D is a dark,low contrast image cause all the gray levels are close to each other and they are all in range of 0-90 based on D histogram. in contrast H wich is the equalized version of D is a brighter picture and has more contrast thus has more quality for human eyes. H has more gray levels and also it's histogram is much more closer to uniform distribution with bins in range of 0-250. H is brighter becuase after equalization as shown in it's histogram there are bins located in range of 150-250 (light areas) which was not the case bnefore histogram equalization.

```
def hist_equalization(img):
  #mn = img.shape[0] * img.shape[1]
  hist_img = calculate_histogram(img)
  mn=np.sum(hist_img)
  pdf = hist_img / mn
# max_intensity = np.max(img)
  max_intensity=255
  out = img.copy()
  cdf = np.cumsum(pdf)
  equalized_gray_levels = np.round_(max_intensity * cdf
```

```
  for i in range(0,out.shape[0]):
    for j in range(0,out.shape[1]):
      out[i,j] = equalized_gray_levels[img[i,j]]

  return out
```

```
H = hist_equalization(D)
hist_H = calculate_histogram(H)
```

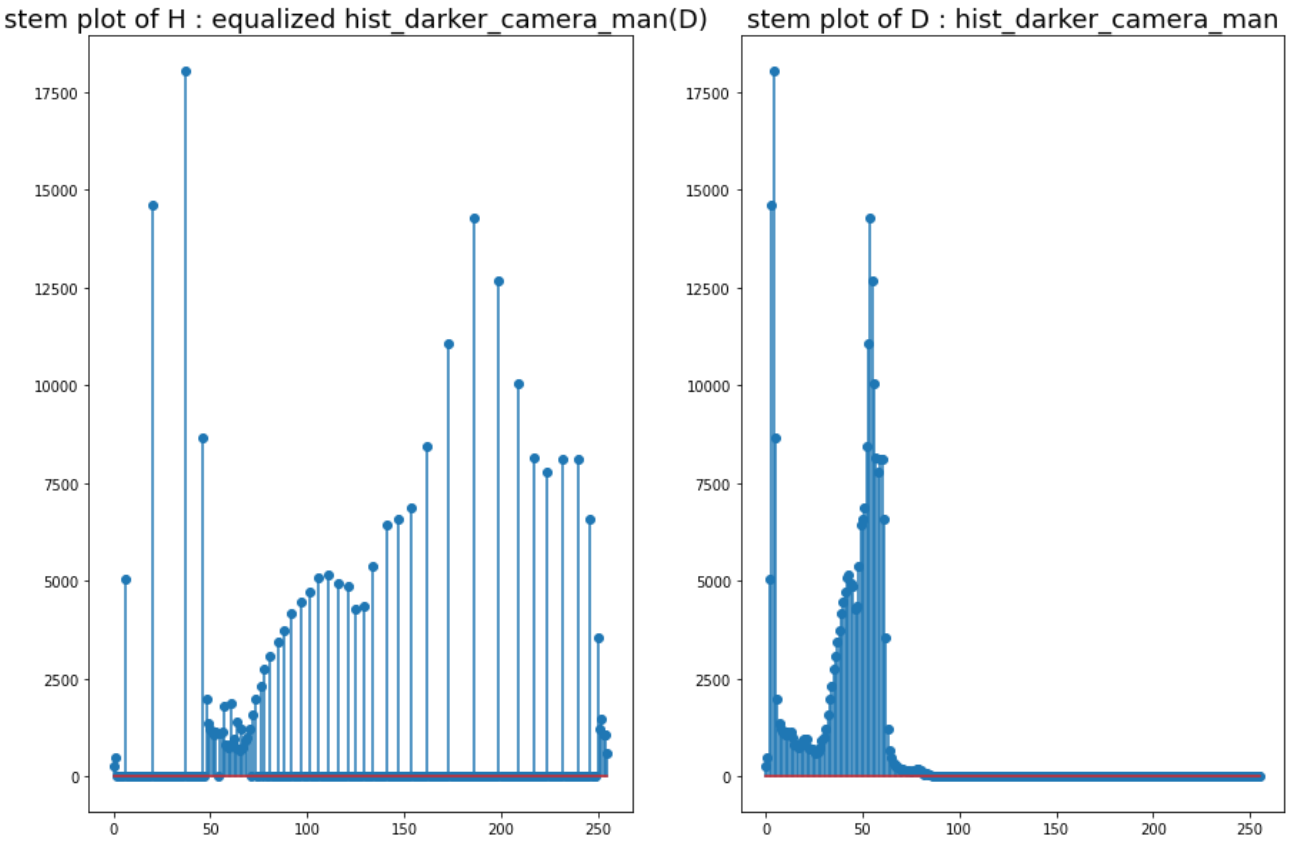
```
fig = plt.figure(figsize=(15, 10))
fig.add_subplot(1, 2, 1)
plt.title('D',fontsize=18)
plt.imshow(D,cmap='gray')

fig.add_subplot(1, 2, 2)
plt.title('H',fontsize=18)
plt.imshow(H,cmap='gray')

plt.show()
```




```
fig = plt.figure(figsize=(15, 10))
fig.add_subplot(1, 2, 1)
```

```
plt.title('stem plot of H : equalized hist_darker_camer
plt.stem(np.linspace(0,255,256),hist_H)

fig.add_subplot(1, 2, 2)
plt.title('stem plot of D : hist_darker_camera_man',for
plt.stem(np.linspace(0,255,256),hist_darker_camera_man)

plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: UserWarning:
  after removing the cwd from sys.path.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
```



2.1.1.4. and 2.1.1.5.

Extract

Histogram Equalization is a computer image processing technique used to improve contrast in images.This method usually increases the global contrast of images.This allows for areas of lower contrast to gain a higher contrast.

Local Histogram Equalization differs from ordinary histogram equalization in the respect that the local method computes several histograms, each corresponding to a distinct section of the image.It is therefore suitable for improving the local contrast and enhancing the definitions of edges in each region of an image. it computes global technique for each patch(window).

Implementation

local_hist_equlization function implements local histogram equalization. first it divides image to patches based on window size(input) and then apply global histogram equalization bt calling hist_equalization function from previous part.

Results

In H(global requalized D) areas of lower contrast gained a higher contrast.for example we can see more details in camera man's black coat or more details in the ground. in L (local requalized D), it worked for improving the local contrast and enhancing the definitions of edges in each region of an image.

histogram equalization enhanced the contrast of the whole image. local histogram equalization enhanced many image details by taking different transformation of the same gray level at different places in the original image.

However, the local histogram equalization process resultsed in unacceptable modification of the original image appearance. the reason for this is that it enhanced cintrast in every window, so local noises and local edges get much more noticable than the original. as you can see in L image each window contains local noises and edges. amount of local noises is changable due window size.(should be between 1/16 to 1/64)

Histogram of L contains more bins than histogram of H, this is because that local histogram equalization applies equalization to each patch of image. on the other

word it extracts more new gray levels than global approach. that why number of bins in L histogram is more than H histogram.

Mostly because of enhancing local noises in local histogram equalization, it is not used that much, instead adaptive local histogram equalization are used more often like CLAHE.

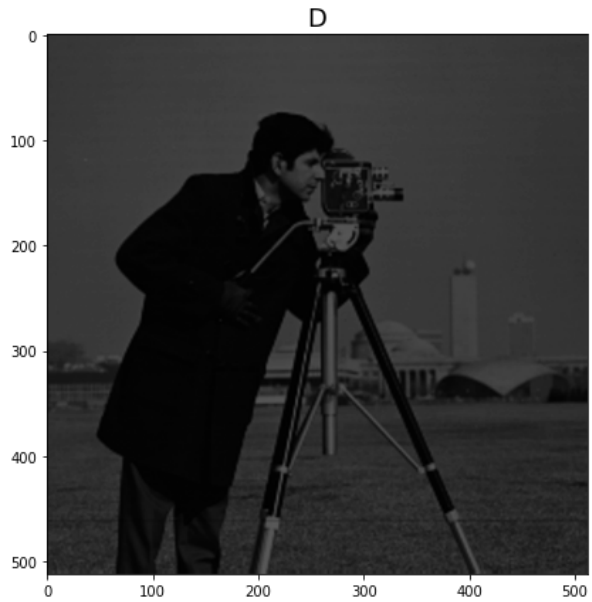
```
D.shape
```

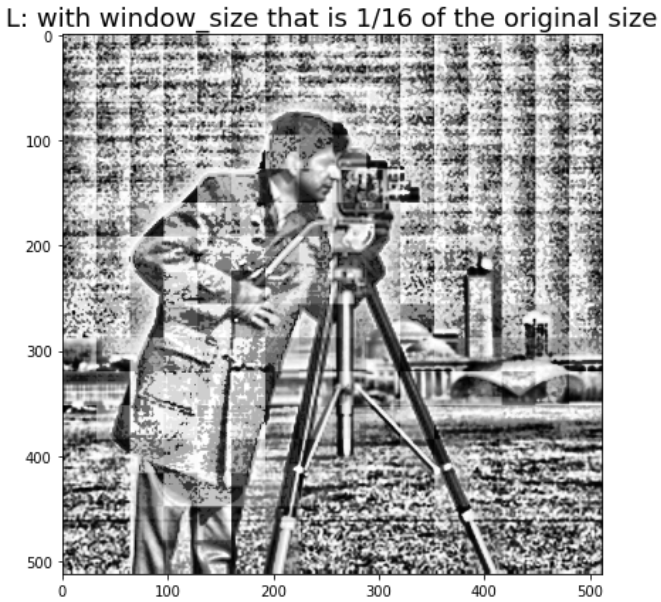
```
(512, 512)
```

```
def local_hist_equlization(img,window_size):
  out = img.copy()
  r_patch = floor(out.shape[0] / window_size)
  c_patch = floor(out.shape[1] / window_size)

  for i in range(0,r_patch):
    for j in range(0,c_patch):
      patch = img[i*window_size:(i+1)*window_size , j*w
      out[i*window_size:(i+1)*window_size , j*window_si

  return out
```

```
L = local_hist_equlization(D,32) # 1/16  of 512
```

```
fig = plt.figure(figsize=(15, 10))
fig.add_subplot(1, 2, 1)
plt.title('D',fontsize=18)
plt.imshow(D,cmap='gray')

fig.add_subplot(1, 2, 2)
plt.title('L: with window_size that is 1/16 of the orig
plt.imshow(L,cmap='gray')

plt.show()
```

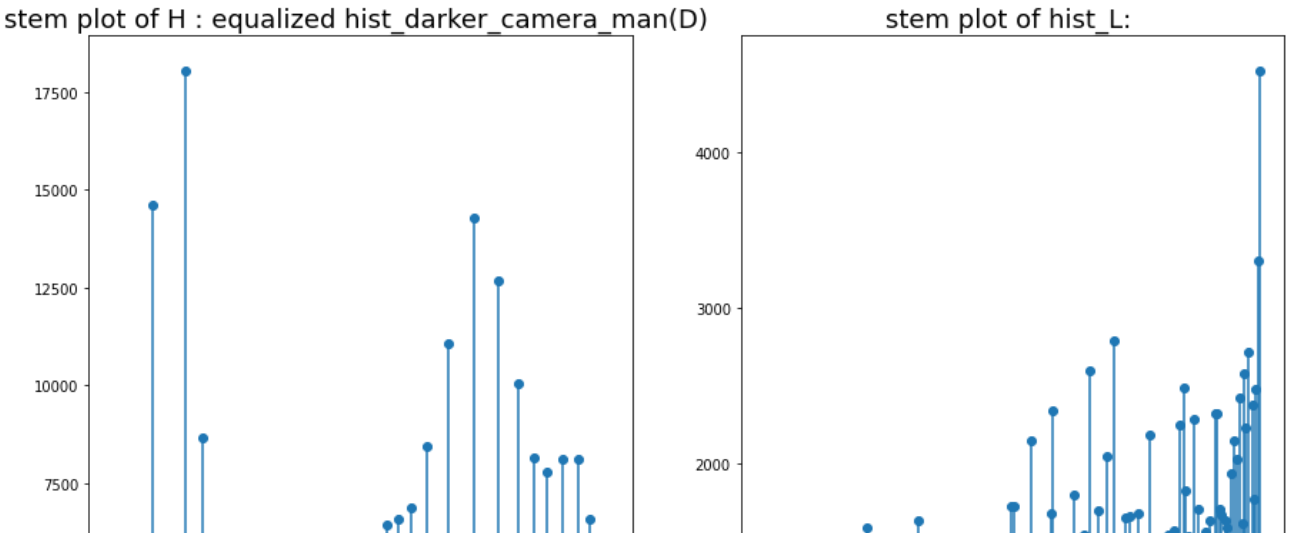





```
hist_L = calculate_histogram(L)
```

```
fig = plt.figure(figsize=(15, 10))
fig.add_subplot(1, 2, 1)
plt.title('stem plot of H : equalized hist_darker_camer
plt.stem(np.linspace(0,255,256),hist_H)

fig.add_subplot(1, 2, 2)
plt.title('stem plot of hist_L: ',fontsize=18)
plt.stem(np.linspace(0,255,256),hist_L)

plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: UserWarning:
  after removing the cwd from sys.path.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
```



2.1.1.6. log transform part

Extract

The general form of the log transformation is:

$$s = c\,log(1 + r)$$

where c is a constant and the role of this multiplier is to make the log-transformed pixel values span the entire range of 256 grayscale levels.

the input intensity values have all been incremented by 1 (r+1). This is because our input values vary from 0 to 255 and the logarithm of 0 is not defined. Secondly, there has been no mention regarding the base of the logarithm. Although conceptually the value of the base doesn't really matter (as long as it's kept same throughout the computation), for all practical purposes, we will assume it to be 10.

The shape of the log curve shows that this transformation maps a narrow range of low intensity values in the input into a wider range of output levels.

transformation of this type to expand the values of dark pixels in image, while compressing the higher-level values. The opposite is true of the inverse log (exponential) transformation.

Implementation

---

log_transform function implements log reansformation. first it calculates maximun gray level in image r_max for computing multiplier c.

The role of the multiplier is to make the log-transformed pixel values span the entire range of 256 grayscale levels available for the output image. The way it's done is by choosing a value of c so that the maximum intensity available in the input image gets mapped to 255 in the output. This means that:

$$255 = c\,log(1 + r_{max})$$

$$c = \frac{255}{log(1 + r_{max})}$$

After computing c, we apply the log transformation formula which was shown earlier on image.

Results

A glance will tell you that the log-transformed image is much more brighter than its counterpart. why is that?

If there are two adjacent pixels with intensities 0 and 15 in the input image, both of them would be almost indistinguishable. Human eyes will not be able to perceive such a subtle change in the grayscale intensity. However, in the log-transformed image, the pixel with the intensity value of 15 gets converted to 127 (which lies in the middle of the grayscale spectrum). This would render it clearly distinguishable from its neighbor, which is still completely black. thats the reasin log_transformes image is brighter and contrast in dark places increased.

For example now we can see the details in camera man's coat like pockets and buttons(darker areas). which shows the fact that log transformation increases contrast in dark areas.

The exact opposite phenomenon takes place at the other parts of the image(brighter places). For example, pixels with intensities of 205 and 255 are mapped to 245 and 255 by the log transform. This means that a significant difference of 50 in the grayscale spectrum has been reduced to a mere gap of 10. So, the log transform essentially magnifies the differences in intensity of pixels in the lower (darker) end of

the grayscale spectrum at the cost of diminishing differences at the higher (brighter) end

For example we have a decrease ofcontrast in the background in brighter areas. in log_tranformed_D we can not specify white buldings behind of camera man from sky clearly whereas befor log tranformation white building was compeletly distinguishable from sky.

By comparing histograms before and after log transform we can see that after transformation, bins from dark gray levels now are spread in much bigger range of gray levels. from 0-30 to 0-160 which means more gary levels for dark places and more contrast in dark areas. also bins in brighter gray levels 160-200 are pretty compressed and close to each other that means deacrese of contrast in bright places.

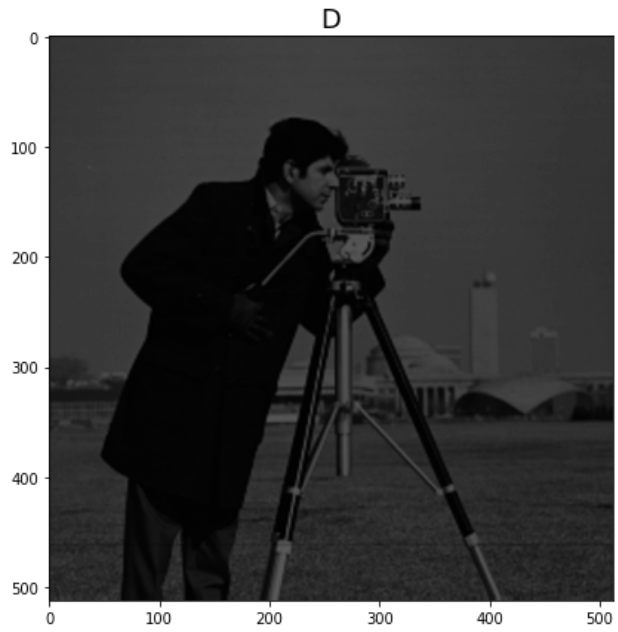
```
def log_transform(img):  # s = c log(r+1)   # 255/log(r
  r_max = np.max(img)
  c = 255/log(r_max +1)
  out = img.copy()
  out = c * np.log(img+1)
  return out.astype('int32')
```

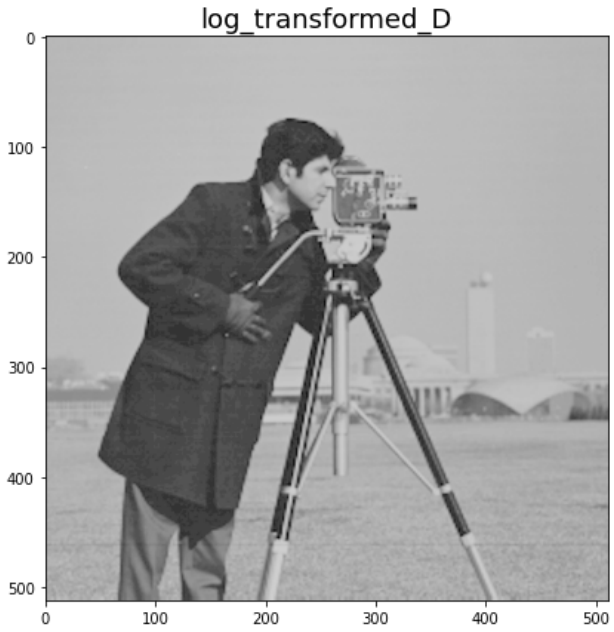
```
log_transformed_D=log_transform(D)
hist_log_transformed_D = calculate_histogram(log_transf
```

```
(log_transformed_D.shape)
```

```
(512, 512)
```

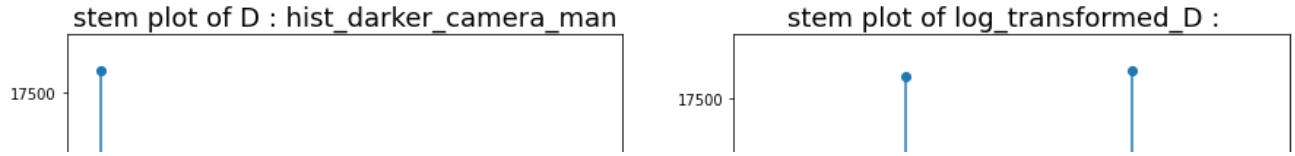
```
fig = plt.figure(figsize=(15, 10))
fig.add_subplot(1, 2, 1)
plt.title('D',fontsize=18)
plt.imshow(D,cmap='gray')

fig.add_subplot(1, 2, 2)
plt.title('log_transformed_D',fontsize=18)
plt.imshow(log_transformed_D,cmap='gray')

plt.show()
```






```
fig = plt.figure(figsize=(15, 10))
fig.add_subplot(1, 2, 1)
plt.title('stem plot of D : hist_darker_camera_man',for
plt.stem(np.linspace(0,255,256),hist_darker_camera_man)

fig.add_subplot(1, 2, 2)
plt.title('stem plot of log_transformed_D :  ',fontsize
plt.stem(np.linspace(0,255,256),hist_log_transformed_D)

plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: UserWarning:
  after removing the cwd from sys.path.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
```



2.1.1.6. inverse log transform(exponential) part

Extract

What it does essentially is the complete opposite of the log transform (hence, it is also named inverse-log transform). While the log transform enhanced the pixels in the lower end of the spectrum, the exponential transform does the same for the pixels at the high intensity end of the spectrum. Mathematically, we have the following:

$$s = c\,(b^r - 1)$$

The exponential transform raises a base value b to the power of the input pixel's intensity value. We subtract so that when the input is 0, the output gets mapped to 0 as well. The constant c plays the same role as in the case of log transform, ensuring that the output lies in the range of 0 to 255.

The inverse log transform maps a narrow range of input intensity values at the higher(brighter) end of the grayscale spectrum to a broader range at the output.

transformation of this type to expand the values of bright pixels in image, while compressing the lower-level values(darker).

Implementation

---

inverse_log_transform function implements log reansformation. first it calculates maximun gray level in image r_max for computing multiplier c.

The role of the multiplier is to make the inverse-log-transformed pixel values span the entire range of 256 grayscale levels available for the output image. The way it's done is by choosing a value of c so that the maximum intensity available in the input image gets mapped to 255 in the output. This means that:

$$255 = c\,(b^r_{max} - 1)$$

$$c = \frac{255}{b_{max}^r - 1}$$

After computing c, we apply the inverse log transformation formula which was shown earlier on image. Also after trying this transformation with b=1 and b=10 and b=1.02, 1.02 had the best result.

Results

A glance will tell you that the inverse-log-transformed image is much more darker than its counterpart. why is that?

Becuase invere-log transformation expands light pixels or in the other word expands range of gray levels in light areas, for exmple assign 230-250 to 127-250. 127 is midrange gray value, means that the spectrum gets darker by expanding rage of bright gray levels.

For example now we can see that the log-transformed image us pretty darker than D. in a way that it is similar to a plain black.

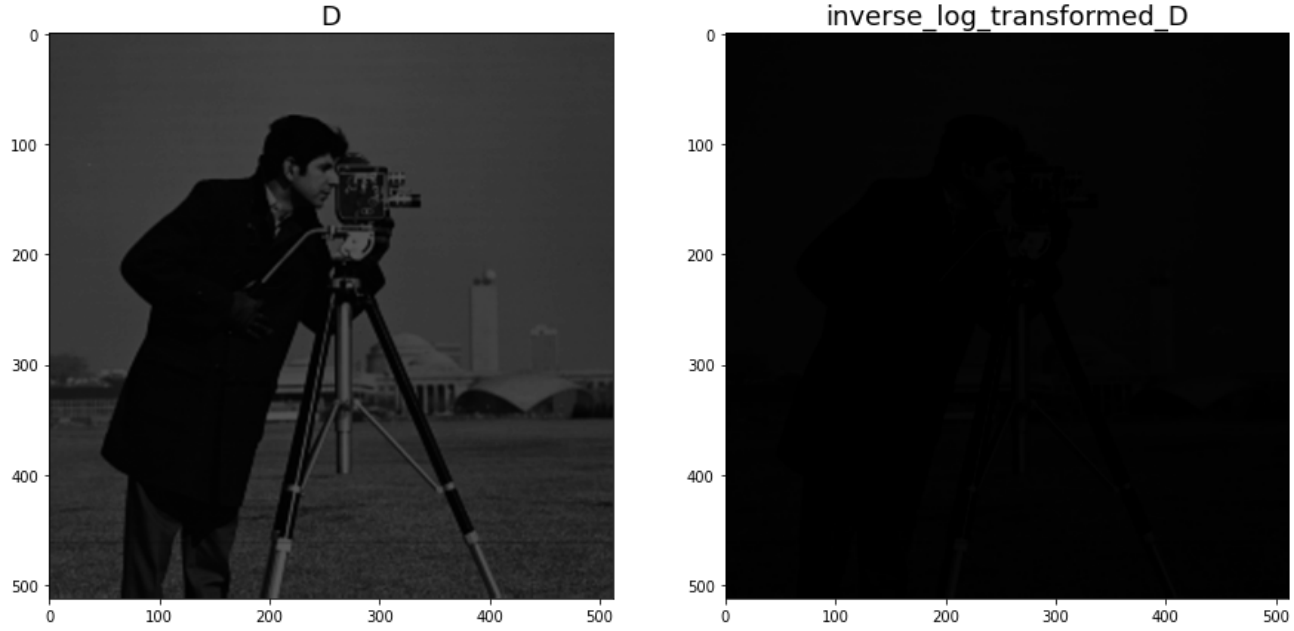
By comparing histograms before and after log transform we can see that after transformation, bins from lighter gray levels now are spreaded out more in dark range on the other hand dark bins are compressed and close to each other. by spreading light bins in darker region and compressing of dark bins, the accumulation of bins is dark region is high.

```
1 def inverse_log_transform(img): # s = c (b^r -1)   c =
2   b = 1.02   #based on https://subscription.packtpub.cc
3
4   r_max = np.max(img)
5   c = 255.0 / (pow(b,r_max) - 1)
6   out = img.copy()
7   out = c * (np.power(b,img) - 1)
8   return out.astype('int32')
```

```
1 inverse_log_transformed_D=inverse_log_transform(D)
2 hist_inverse_log_transformed_D = calculate_histogram(ir
```

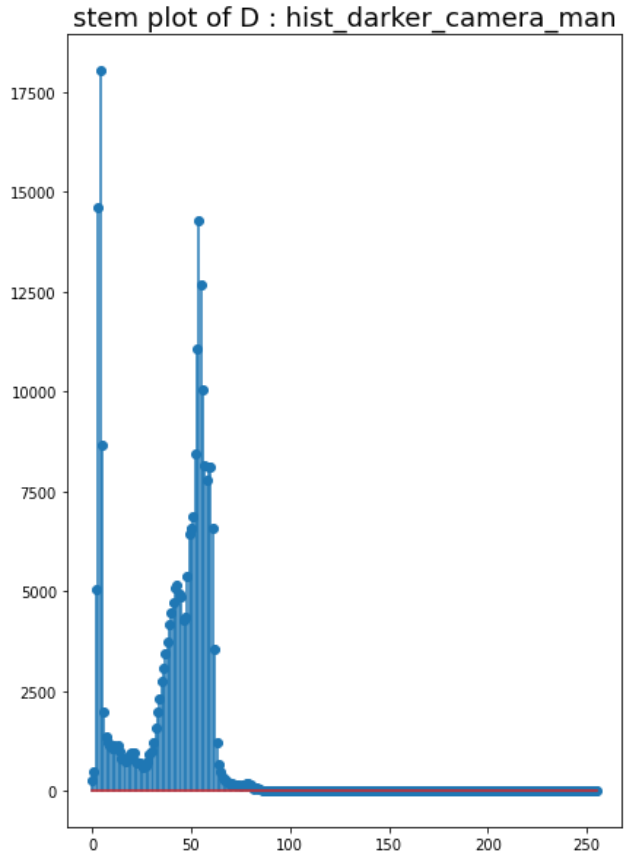
```
1 fig = plt.figure(figsize=(15, 10))
```

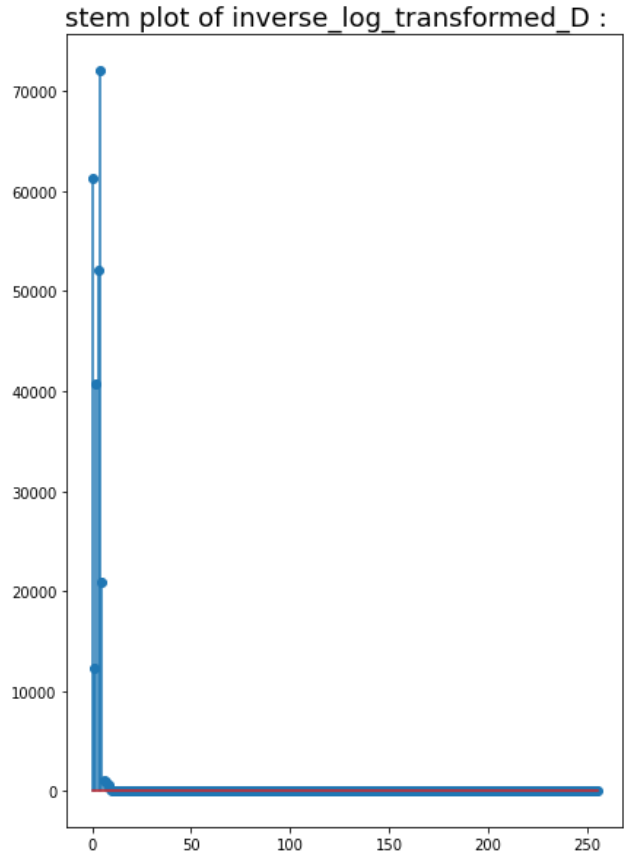
```
fig.add_subplot(1, 2, 1)
plt.title('D',fontsize=18)
plt.imshow(D,cmap='gray')

fig.add_subplot(1, 2, 2)
plt.title('inverse_log_transformed_D',fontsize=18)
plt.imshow(inverse_log_transformed_D,cmap='gray')

plt.show()
```




```
fig = plt.figure(figsize=(15, 10))
fig.add_subplot(1, 2, 1)
plt.title('stem plot of D : hist_darker_camera_man',for
plt.stem(np.linspace(0,255,256),hist_darker_camera_man)

fig.add_subplot(1, 2, 2)
plt.title('stem plot of inverse_log_transformed_D :  ',
plt.stem(np.linspace(0,255,256),hist_inverse_log_transf

plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: UserWarning:
  after removing the cwd from sys.path.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
```





2.1.1.6. Power-Law(gamma) transform part

Extract

Gamma correction is important for displaying images on a screen correctly,This is done because our eyes perceive images in a gamma-shaped curve, whereas cameras capture images in a linear fashion. Power-law transformations have the form:

$$s = c\, r^{\gamma}$$

where c and g are positive constants.where c is a constant and the role of this multiplier is to make the log-transformed pixel values span the entire range of 256(0-255) grayscale levels.

Power-law curves with fractional (gamma<1)values of gamma map a narrow range of dark input values into a wider range of output values, with the opposite (gamma>1) a narrow range of bright input values into a wider range of output values

Implementation

---

power_low function implements gamma transformation. first it divides gray levels by 255 to normalize values between 0-1 to prevent exploding the intensity values.(by powering 0-255 values to gamma,the intensity values explode)

This normalized to values to the power of gamma, are mutiplied by 255 to be in range 0-255 again.

Results

D ia dark image and as you can see gamma values less than 1 have better results. gamma transformed images with values 1.7,2.5,3 are low contrast and gamma=3 is completly balck with contrast = 0, means that gamma>1 is not suitable for D.

For gamma<1, condition is better. the best gamma value is 0.5, beacuase we can see the detail in black areas like pockets and buttons of camera man's black coat. also the white buldings are distingushable from sky in background. gamma=0.25 is washed out espicially in background and gamma =0.75,0.9,1 are dark and low contrast. gamme= 0.5 has the best result.

```
def power_low(img,gamma=1):
  out = img.copy()
  out =np.array(255*(img / 255) ** gamma, dtype = 'uint
  return out
```

```
def show_img(*args, figsize=10, is_gray=True, title=Nor

  if isinstance(figsize, int):
    figsize = (figsize, figsize)
  images = args[0] if type(args[0]) is list else list(a
  cmap=None
  if not is_gray:
    images = list(map(lambda x: cv2.cvtColor(x, cv2.COL
```

```
  else:
    cmap = 'gray'
  plt.figure(figsize=figsize)
  for i in range(1, len(images)+1):
    plt.subplot(1, len(images), i)
    if title is not None:
      plt.title(title[i-1], fontsize=fontsize)

    plt.imshow(images[i-1], cmap=cmap)
    plt.axis('off')
```

```
power_law_transformed_D = []
gamma = [0.25, 0.5, 0.75, 0.9,1,1.7,2.5,3]
for i in gamma:
  power_law_transformed_D.append(power_low(D, gamma=i))
show_img(power_law_transformed_D, figsize=25, title=gam
```

2.1.2

Extract

Histogram Equalization is a technique that changes histogram distribution to uniform distribution. histogram with uniform distribution has bins in almost all gray level values and length of all bins is almost the same. meaning that this picture has almost all gray levels.

Histogram Equalization on a picture increases gray levels in a picture to close the distribution to uniform distribution thus the contrast and quality of image increases. it is a technique for boosting contrast in low contrast images.

Implementation

---

hist_equalization function implements histogram equalization. first it calculates histogram of an image by using calculate_histogram function from previous part then sums up all bins in histogram to get the total number of gray levels in image(mn). Then the normalized hisftogram or probability density function(pdf) is calculated by deviding histogram to mn(total number of gray levels in image).

Then the commulative density function for histogram is calculated by using np.cumsum on pdf. by multiplying max intensity value (for mapping bins) in picture by cdf we have the equalized version of histogram.

Results

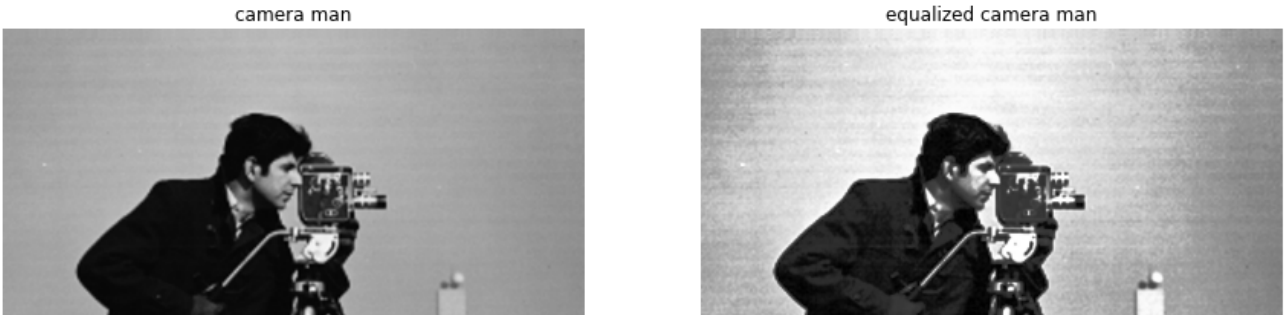
---

camera_man_equalized is the equalized version of camera_man by performing hist_equalization(camera_man). we can see more details in camera_man_equalized like buttons and pockets in camera man's balck coat which was not the case in camera_man image. also the white buildins are more distiguishable from sky in camera_man_equalized than camera_man. this bosst of quality and details is because of contrast increasment of histogram equlization on camera_man.

Range of bins in camera man histogram is 0-200 in contrast this range is 0-255 after histogram equlization. means that after equlization, image has more gray levels and more contrast.

```
1 camera_man_equalized=hist_equalization(camera_man_img)
2 hist_camera_man_equalized = calculate_histogram(camera_
```

```
1 show_img(camera_man_img, camera_man_equalized, title=['
```




```
fig = plt.figure(figsize=(15, 10))
fig.add_subplot(1, 2, 1)
plt.title('stem plot of hist_camera_man',fontsize=18)
plt.stem(np.linspace(0,255,256),hist_camera_man)

fig.add_subplot(1, 2, 2)
plt.title('stem plot of equalized_camera_man',fontsize=
plt.stem(np.linspace(0,255,256),hist_camera_man_equaliz

plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: UserWarning:
  after removing the cwd from sys.path.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
```

2.1.3

```
def imadjust(img,low_out=0,high_out=1,gamma=1):
  low_in = np.min(img)
  high_in = np.max(img)
  out =(((img - low_in) / (high_in - low_in))**gamma) *
  return out.astype('uint8')
```

```
in_out_bounds=[(24,64),(32,196),(16,232),(0,255)]
imadjs = []
titles=[]
for (l,h) in in_out_bounds:
  #titles.append(f'low boundray {l} boundry {h}')
  imadjs.append(imadjust(camera_man_img,l,h))

```

```
from google.colab.patches import cv2_imshow
cv2_imshow(imadjs[0])
```

```
1 cv2_imshow(imadjs[1])
```

```
1 cv2_imshow(imadjs[2])
```

```
cv2_imshow(imadjs[3])
```

```
show_img(camera_man_equalized, title=['equalized camera
```

2.2

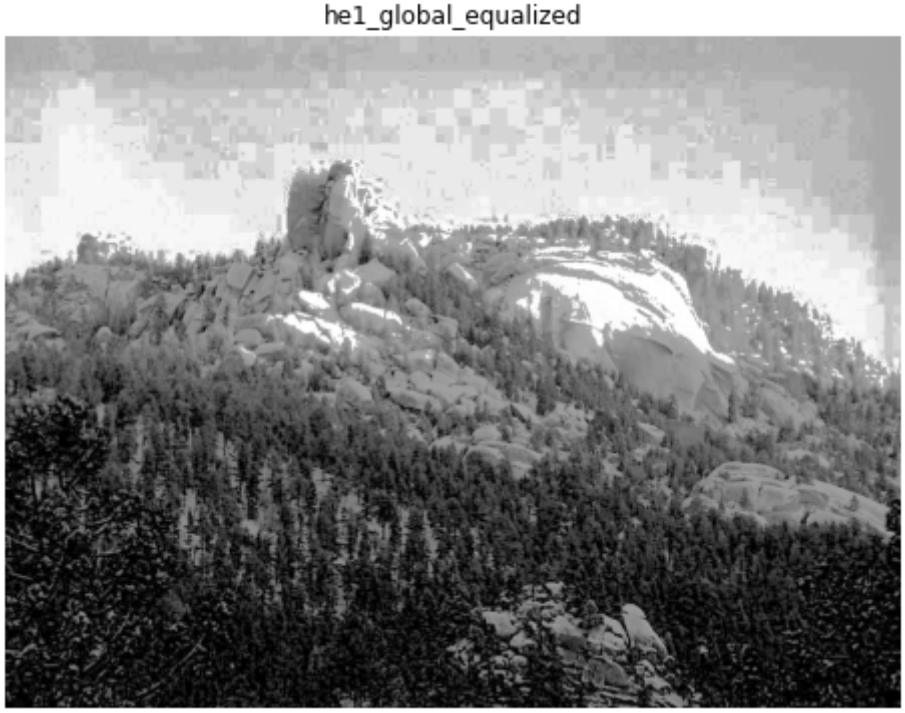
2.2.1

```
1 he1 = cv2.imread('HE1.jpg',cv2.IMREAD_GRAYSCALE)
2 he2 = cv2.imread('HE2.jpg',cv2.IMREAD_GRAYSCALE)
3 he3= cv2.imread('HE3.jpg',cv2.IMREAD_GRAYSCALE)
4 he4= cv2.imread('HE4.jpg',cv2.IMREAD_GRAYSCALE)
```

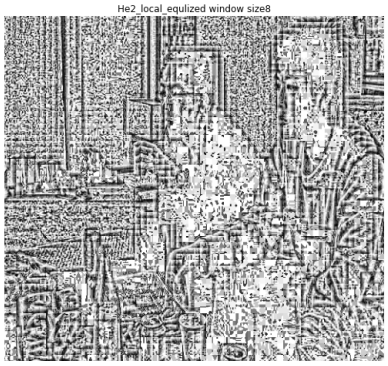
```
1 show_img(he1,he2,he3,he4 ,title=['he1','he2','he3','he4
```

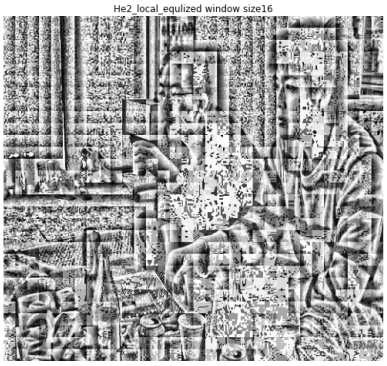

```
1 he1_local_equlization_list=[]
2 titles=[]
3 for w in [8,16,32]:
4   he1_local_equlized= local_hist_equlization(he1,w)
5   he1_local_equlization_list.append(he1_local_equlized
6   titles.append(f'He1_local_equlized window size{w}')
7 show_img(he1_local_equlization_list,title=titles, figsi
```

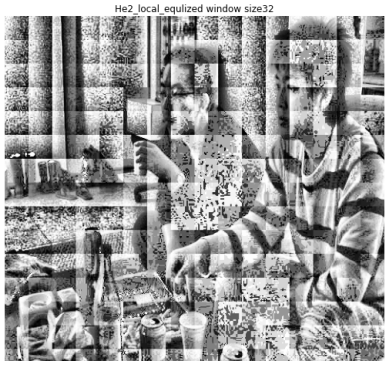

```
1 he1_global_equalized=hist_equalization(he1)
2 show_img(he1_global_equalized, title=['he1_global_equal
```



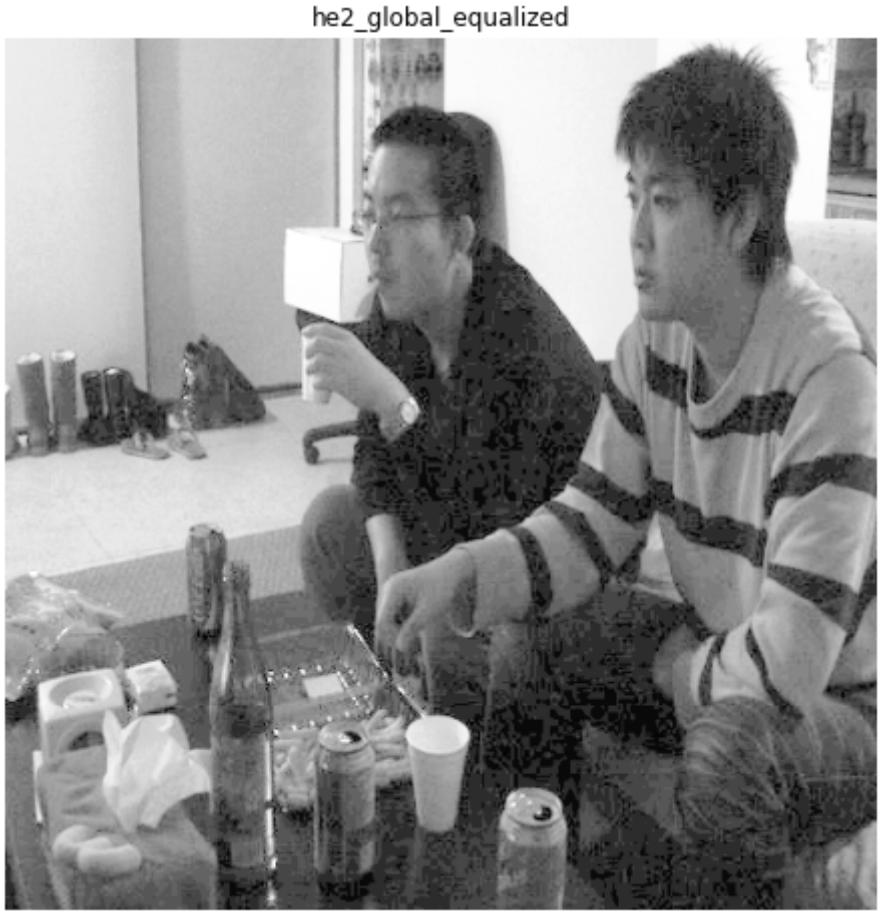

```
1 he2_local_equlization_list=[]
2 titles=[]
3 for w in [8,16,32]:
4    he2_local_equlized= local_hist_equlization(he2,w)
5    he2_local_equlization_list.append(he2_local_equlized
6    titles.append(f'He2_local_equlized window size{w}')
7 show_img(he2_local_equlization_list,title=titles, figsi
```



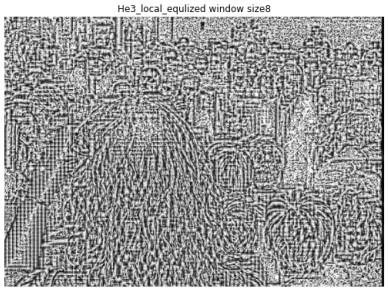


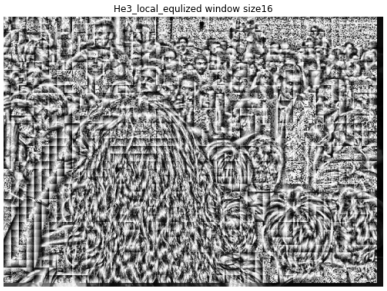



```
he2_global_equalized=hist_equalization(he2)
show_img(he2_global_equalized, title=['he2_global_equal
```



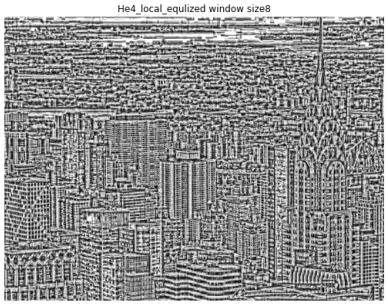

```
he3_local_equlization_list=[]
titles=[]
for w in [8,16,32]:
   he3_local_equlized= local_hist_equlization(he3,w)
   he3_local_equlization_list.append(he3_local_equlized
   titles.append(f'He3_local_equlized window size{w}')
show_img(he3_local_equlization_list,title=titles, figsi
```

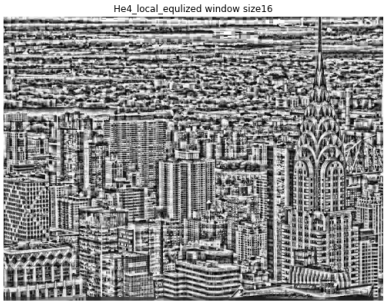





```
1 he3_global_equalized=hist_equalization(he3)
2 show_img(he3_global_equalized, title=['he3_global_equal
```


```
1 he4_local_equlization_list=[]
2 titles=[]
3 for w in [8,16,32]:
4   he4_local_equlized= local_hist_equlization(he4,w)
5   he4_local_equlization_list.append(he4_local_equlized
6   titles.append(f'He4_local_equlized window size{w}')
7 show_img(he4_local_equlization_list,title=titles, figsi
```






```
1 he4_global_equalized=hist_equalization(he4)
2 show_img(he4_global_equalized, title=['he4_global_equal
```

Colab paid products - Cancel contracts here