Digital Image Processing

Assignment 1

School of Computer Science & Applied Mathematics
University of the Witwatersrand

Author: **Sibusiso Mgidi**

February 1, 2021

Contents

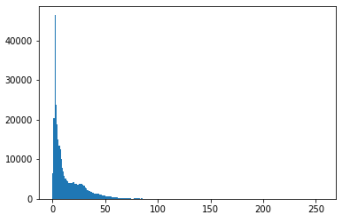
1 Low light image contrast enhancement

1.1 Contrast stretching

Display input, output and plots

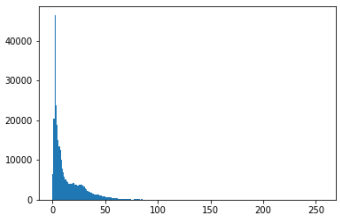
Original image

Original image histogram



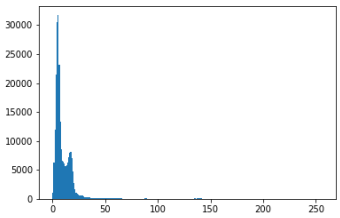
Transformed image

Transformed image histogram



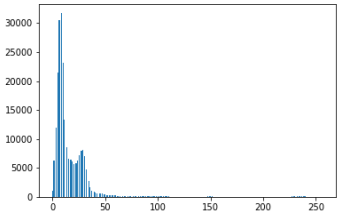
Original image

Original image histogram

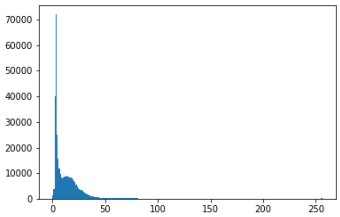


Transformed image

Transformed image histogram

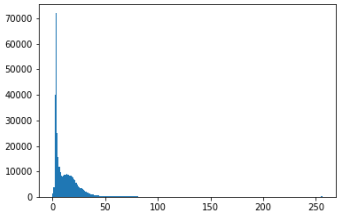


Original image



Original image histogram

Transformed image



Transformed image histogram

Discussion and comments on the findings above:

Contrast stretching is a basic image enhancing technique that aims to increase the contrast in a given image. In the figure above we applied minimum and maximum stretching. Using this method in the images above, yield close to no difference in the transformed image. Only one image slightly increase intensity level after applying contrast stretching. Given the results above, we can conclude that contrast stretching does not improve the results much.

Source code:

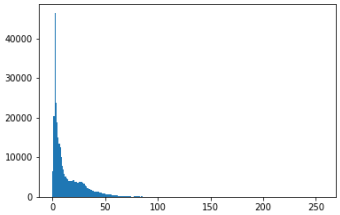
```
from skimage import io
import numpy as np
import matplotlib.pyplot as plt
from skimage.color import rgb2gray

# reading and converting color image from RGB to grasyscale
    image
img = rgb2gray(io.imread(.../))

# Plot the original image
plt.hist(img.ravel(), 256, [0, 256])
plt.savefig("1_1__original_road_low_1_hist.png")
plt.show()

# Contrast stretching function
```

```
def contrast_stretching(img):
    # Initialize the array to zeros in order to store the
    # transformed image
    output_img = np.zeros(img.shape).astype("uint8")
    # Get the minimum and maximum pixel in an image
    pixel_min = np.min(img)
    pixel_max = np.max(img)

    # Loop over the image and apply contrast stretching
    for row in range(img.shape[0]):
        for col in range(img.shape[1]):
            output_img[row][col] = 255 * ((img[row][col] -
               pixel_min)/(pixel_max - pixel_min))

    # Transformed image
    return output_img

# Call the contract stretching function
image = contrast_stretching(img)

# Histogram plot of a tranformed image
plt.hist(image.ravel(), 256, [0, 256])
plt.savefig("1_1_transformed_road_low_1_hist.png")
plt.show()

# Plot the original image in comaprison with the equalized
   image
fig = plt.figure(figsize=(20,20))

# Show original image
ax1 = fig.add_subplot(1,2,1)
ax1.set_title("Original Image")
plt.imshow(img, cmap='gray')

# Histogram equalized images
ax2 = fig.add_subplot(1,2,2)
ax2.set_title("Transformed Image")
plt.imshow(image, cmap='gray')
plt.show(block=True)

# Save the image on a local machine
io.imsave('.../', image)
```

1.2 Histogram equalization

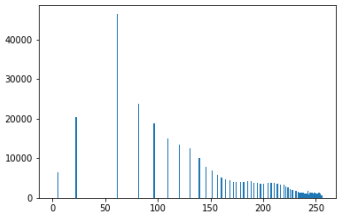
Original image

Original image histogram



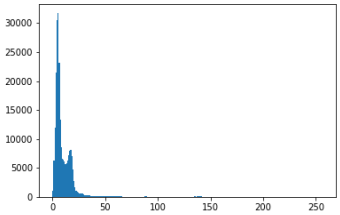
Transformed image

Transformed image histogram



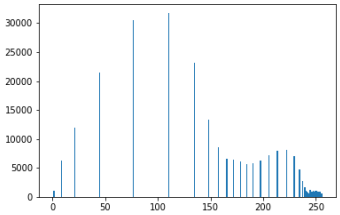
Original image

Original image histogram

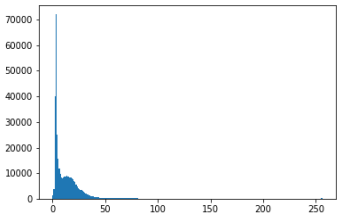


Transformed image

Transformed image histogram

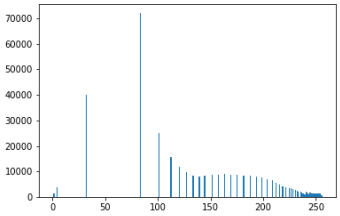


Original image



Original image histogram

Transformed image



Transformed image histogram

Discussion and comments on the findings above:

Histogram equalization is an image processing technique that manipulates the image with an intention to increase its contrast. This is achieved by efficiently scattering the most commonly encountered intensity level. Looking at the results above, it is evident that histogram equalization yield better results as compared to contrast stretching. Although the results present significant improvement over contrast stretching, most features are lost when applied to image with slightly higher pixel values.

Source code:

```
from skimage import io
import numpy as np
import matplotlib.pyplot as plt
from skimage.color import rgb2gray

# reading and converting color image from RGB to grasyscale
    image
img = rgb2gray(io.imread('.../'))

# Plot the original image
plt.imshow(img, aspect = 'auto', cmap = 'gray')

# Function for Ploting an original image histogram
def plot_histogram(img):
    plt.hist(img.ravel(),256,[0,256])
```

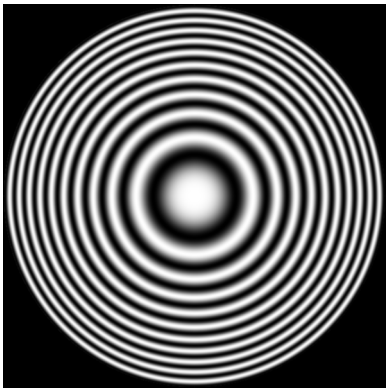
```
    plt.savefig("original_road_low_2_hist.png")
    plt.show()

# Original image plot
plot_histogram(img)

# Total number of of rows and columns in an image
n = img.shape[0] * img.shape[1]
[row,column] = img.shapeqx

# One dimensional array of length 256 with values from 0 - 255
image_matrix = np.arange(0,256,1)

# Initialize the array to zeros
image_matrix[:] = 0
for i in range(0, row):
    for j in range(0, column):
        image_matrix[img[i,j]] += 1
# Frequency
image_matrix

# Histogram
pr = image_matrix/(n) #pdf

# calculate cdf(cummulative density function) of gray scale
   image
cdf_sum = []
cum_sum = 0
for pixel in pr:
    cum_sum = cum_sum + pixel
    cdf_sum.append(cum_sum)

# cdf * (L- 1)
L = 256
cdf_sum_with_L = []
for value in cdf_sum:
    element = np.round(value * (L - 1))
    cdf_sum_with_L.append(element)

#Initialize the array to zeros in order to store the
   tranformed image
outputImg = np.zeros(img.shape).astype('uint8')
for row in range(img.shape[0]):
    for column in range(img.shape[1]):
        i = img[row][column]
        outputImg[row][column]=cdf_sum_with_L[i]
```

```
# Function to ploting and saving a tranformed image histogram
def plot_histogram(outputImg):
    plt.hist(outputImg.ravel(),256,[0,256])
    plt.savefig("transformed_road_low_2_hist.png")
    plt.show()

# Plot tranformed image histogram
plot_histogram(outputImg)

# Plot the original image in comaprison with the equalized
  image
fig = plt.figure(figsize=(20,20))

# Show original image
ax1 = fig.add_subplot(1,2,1)
ax1.set_title("Original Image")
plt.imshow(img, cmap='gray')

# Histogram equalized images
ax2 = fig.add_subplot(1,2,2)
ax2.set_title("Equalized Image")
plt.imshow(outputImg, cmap='gray')

io.imsave('../', outputImg)
```

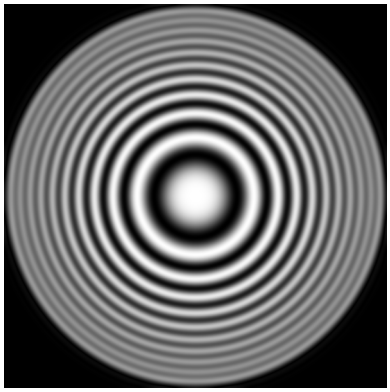
2 Spatial and frequ ency domain filtering

2.1 Low-pass filtering

Original image



Low pass filtered image



Discussion and comments on the findings above:

Low pass filter is used to remove high frequency components and keep the low frequency components in an image. The results become blurrier after applying low pass filter since low light images reflects raw features of an image. In the image above the frequency cut off was low which yield a blurrier image as an output.

Source code:

```
from skimage import io
import numpy as np
import matplotlib.pyplot as plt
from skimage.color import rgb2gray

# Reading and converting color image from RGB to grasyscale
    image
img = rgb2gray(io.imread(r'C:\Users\Sibusiso Mgidi\Desktop\
    Digital Image Processing\Assignment 1\imgs\zoneplate.tif'))

# Creating a low pass filter mask
P, Q = img.shape

# Setting up the range of variables
u = np.arange(P)
v = np.arange(Q)

# Computing indices to use in meshgrid
idx_x = np.where( u > P / 2 )
ind_y = np.where( v > Q / 2 )
u[idx_x] = u[idx_x] - P
v[idx_x] = v[idx_x] - Q

# Create a mask grid out of an image of u and v frequency.
V,U = np.meshgrid(v,u)

# Compute the distance from the center of the frequency
    rectangle.
D = np.sqrt(V**2+U**2)
# Compute the DFT, F(u,v) of the frequency rectangle
dft = np.fft.fft2(img)
img_shift = np.real(np.fft.fftshift(dft))

# Initialize order value
n = 3

# Computing cutoff frequency
```

```
D0 = 0.05 * Q;

# Design the butterworth filter H(u,v)
butterworth_filter = 1/(1 + (D/D0)**(2*n))

# Compute G(u,v) from butterworth filter, H(u,v) and DTF, F(u,
    v) : G(u,v) = H(u,v) * F(u,v)
G = butterworth_filter * dft

# Compute the DFT inverse DFT in order to generate filtered
    image
filtered_img = np.real(np.fft.ifft2(G))

# Obtaining the absolute value of all each element
img_out = np.abs(filtered_img)

# Plot the original image in comaprison with the transformed
    image
fig = plt.figure(figsize=(20,20))

# Show original image
ax1 = fig.add_subplot(1,2,1)
ax1.set_title("Original Image")
plt.imshow(img, cmap='gray')

# Transformed image
ax2 = fig.add_subplot(1,2,2)
ax2.set_title("Low Pass Filter")
plt.imshow(img_out, cmap='gray')

# Saving the tranformed image on a local machine
io.imsave(r'C:\Users\Sibusiso Mgidi\Desktop\Digital Image
    Processing\Assignment 1\imgs\2_1_zoneplate_tranformed.png',
    np.uint8(img_out))
```

2.2 High-pass filtering

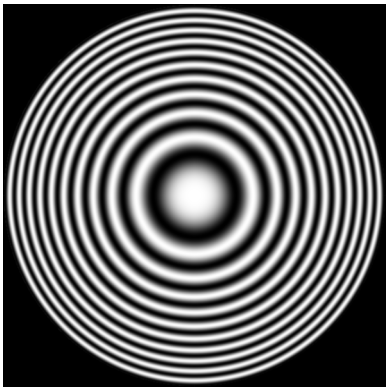
Discussion and comments on the findings above:

High pass filter remove the low frequency components from an image and keeps high frequency components. As we can see from the results achieved after applying high pass filter on the original image, the quality of the image is hardly recognized. This was achieved by subtracting the 1 from the low pass filter.

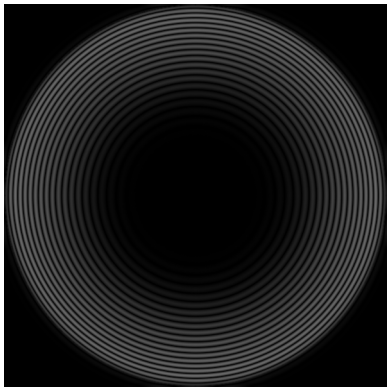
Source code:

```
from skimage import io
```

Original image



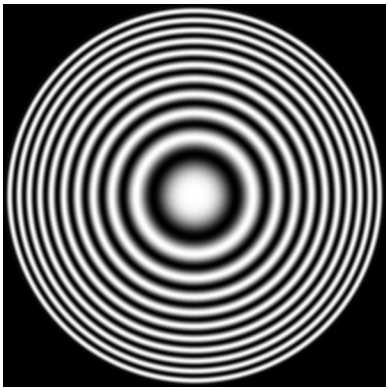
High pass filtered image



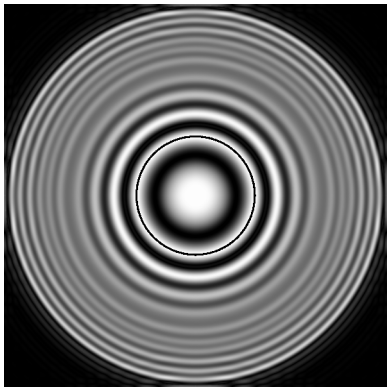
```
import numpy as np
import matplotlib.pyplot as plt
from skimage.color import rgb2gray

# reading and converting color image from RGB to grasyscale
    image
img = rgb2gray(io.imread(r'C:\Users\Sibusiso Mgidi\Desktop\
    Digital Image Processing\Assignment 1\imgs\zoneplate.tif'))

# Creating a low pass filter mask
P, Q = img.shape

# Setting up the range of variables
u = np.arange(P)
v = np.arange(Q)

# Computing indices to use in meshgrid
idx_x = np.where( u > P / 2 )
ind_y = np.where( v > Q / 2 )
u[idx_x] = u[idx_x] - P
v[idx_x] = v[idx_x] - Q

# Create a mask grid out of an image of u and v frequency.
V,U = np.meshgrid(v,u)

# Compute the distance from the center of the frequency
    rectangle.
D = np.sqrt(V**2+U**2)

# Compute the DFT, F(u,v) of the frequency rectangle
dft = np.fft.fft2(img)
img_shift = np.real(np.fft.fftshift(dft))
```

```
# Initialize order value
n = 3

# Computing cutoff frequency
D0 = 0.05 * Q;
#D0 = 70;

# Design the butterworth filter H(u,v)
bt_filter = 1/(1 + (D/D0)**(2*n))

# Compute high pass filter
highpass_filter = 1 - bt_filter

# Compute G(u,v) from butterworth filter, H(u,v) and DTF, F(u,
    v) : G(u,v) = H(u,v) * F(u,v)
G = highpass_filter * dft

# Compute the DFT inverse DFT in order to generate filtered
    image
filtered_img = np.real(np.fft.ifft2(G))

# Obtaining the absolute value of all each element
tranformed_img = np.abs(filtered_img)

# Plot the original image in comparison to the transformed
    image
fig = plt.figure(figsize=(20,20))

# Show original image
ax1 = fig.add_subplot(1,2,1)
ax1.set_title("Original Image")
plt.imshow(img, cmap='gray')

# Transformed image
ax2 = fig.add_subplot(1,2,2)
ax2.set_title("High Pass Filter")
plt.imshow(tranformed_img, cmap='gray')

# Saving the tranformed image on a local machine
io.imsave(r'C:\Users\Sibusiso Mgidi\Desktop\Digital Image
    Processing\Assignment 1\imgs\2_2_zoneplate_tranformed.png',
    np.uint8(tranformed_img))
```

Original image



Band-reject filtered image



2.3 Band-reject filtering

Discussion and comments on the findings above:

The output of this filter is quite similar to the butterworth high pass filter however the background of the image is not removed.

Source code:

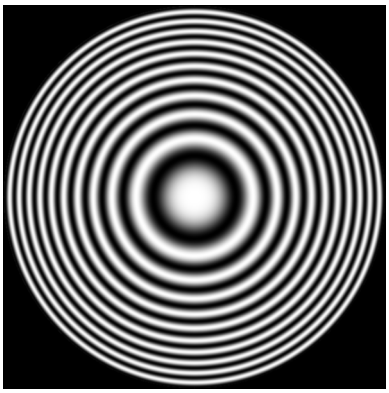
```
from skimage import io
import numpy as np
import matplotlib.pyplot as plt
from skimage.color import rgb2gray

# Reading and converting color image from RGB to grasyscale
    image
img = rgb2gray(io.imread(r'C:\Users\Sibusiso Mgidi\Desktop\
    Digital Image Processing\Assignment 1\imgs\zoneplate.tif'))

# Creating a low pass filter mask
P, Q = img.shape

# Setting up the range of variables
u = np.arange(P)
v = np.arange(Q)

# Computing indices to use in meshgrid
idx_x = np.where( u > P / 2 )
ind_y = np.where( v > Q / 2 )
u[idx_x] = u[idx_x] - P
v[idx_x] = v[idx_x] - Q

# Create a mask grid out of an image of u and v frequency.
```

```
V,U = np.meshgrid(v,u)

# Compute the distance from the center of the frequency
    rectangle.
D = np.sqrt(V**2+U**2)

# Compute the DFT, F(u,v) of the frequency rectangle
dft = np.fft.fft2(img)
img_shift = np.real(np.fft.fftshift(dft))

# Initialize order value
n = 3

# Computing cutoff frequency: D0 (center frequency of the
    reject band)
D0 = 0.05 * Q

# Initialize band width: (W)
W = 15

# Design the butterworth band reject filter HBBR(u,v)
HBBr = 1 / (1 + ((D * W)/(D**2 - D0**2))**(2*n))

# Compute G(u,v) from butterworth filter, H(u,v) and DTF, F(u,
    v) : G(u,v) = H(u,v) * F(u,v)
G = HBBr * dft

# Compute the DFT inverse DFT in order to generate filtered
    image
filtered_img = np.real(np.fft.ifft2(G))

# Obtaining the absolute value of all each element
tranformed_img = np.abs(filtered_img)

# Plot the original image in comparison to the transformed
    image
fig = plt.figure(figsize=(20,20))

# Show original image
ax1 = fig.add_subplot(1,2,1)
ax1.set_title("Original Image")
plt.imshow(img, cmap='gray')

# Show transformed image
ax2 = fig.add_subplot(1,2,2)
ax2.set_title("Band Reject Filter")
```

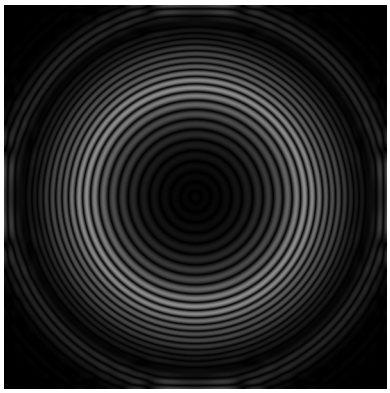
```
plt.imshow(tranformed_img, cmap='gray')

# Saving the tranformed image on a local machine
io.imsave(r'C:\Users\Sibusiso Mgidi\Desktop\Digital Image
    Processing\Assignment 1\imgs\2_3_zoneplate_tranformed.png',
    np.uint8(tranformed_img))
```

2.4 Band-pass filtering

Original image



Band-pass filtered image



Discussion and comments on the findings above:

This is the inverse of band reject filter which transfers out of the frequencies within a spectrum.

Source code:

```
from skimage import io
import numpy as np
import matplotlib.pyplot as plt
from skimage.color import rgb2gray

# reading and converting color image from RGB to grasyscale
    image
img = rgb2gray(io.imread(r'C:\Users\Sibusiso Mgidi\Desktop\
    Digital Image Processing\Assignment 1\imgs\zoneplate.tif'))

# Creating a low pass filter mask
P, Q = img.shape

# Setting up the range of variables
u = np.arange(P)
v = np.arange(Q)

# Computing indices to use in meshgrid
idx_x = np.where( u > P / 2 )
ind_y = np.where( v > Q / 2 )
u[idx_x] = u[idx_x] - P
v[idx_x] = v[idx_x] - Q
```

```
# Create a mask grid out of an image of u and v frequency.
V,U = np.meshgrid(v,u)

# Compute the distance from the center of the frequency
    rectangle.
D = np.sqrt(V**2+U**2)

# Compute the DFT, F(u,v) of the frequency rectangle
dft = np.fft.fft2(img)
img_shift = np.real(np.fft.fftshift(dft))

# Initialize order value
n = 3

# Computing cutoff frequency: D0 (center frequency of the
    reject band)
D0 = 0.04 * Q

# Initialize band width: (W)
#W = 15
W = 15

# Design the butterworth band reject filter HBBR(u,v)
HBBr = 1 / (1 + ((D * W)/(D**2 - D0**2))**(2*n))

# Computing a band pass filter from butterworth band reject
    HBBR(u,v)
HBP = 1 - HBBr

# Compute G(u,v) from butterworth filter, H(u,v) and DTF, F(u,
    v) : G(u,v) = H(u,v) * F(u,v)
G = HBP * dft

# Compute the DFT inverse DFT in order to generate filtered
    image
filtered_img = np.real(np.fft.ifft2(G))

# Obtaining the absolute value of all each element
tranformed_img = np.abs(filtered_img)

# Plot the original image in comparison to the transformed
    image
fig = plt.figure(figsize=(20,20))

# Show original image
```

```
ax1 = fig.add_subplot(1,2,1)
ax1.set_title("Original Image")
plt.imshow(img, cmap='gray')

# Show transformed image
ax2 = fig.add_subplot(1,2,2)
ax2.set_title("Band Pass Filter")
plt.imshow(tranformed_img, cmap='gray')

# Saving the tranformed image on a local machine
io.imsave(r'C:\Users\Sibusiso Mgidi\Desktop\Digital Image
    Processing\Assignment 1\imgs\2_4_zoneplate_tranformed.png',
    np.uint8(tranformed_img))
```