Labsheet 6
Image Transforms,Histogram Equalization and Specification,Filtering

Ragja Palakkadavath
(SC18M003)

Sub: Image and Video Processing Lab
Department: Mathematics(MLC)
Date of Lab sheet: September 19, 2018
Date of Submission: September 23, 2018

---

1. Log Transformation (use cameraman,horizontal,vertical images)

s = Clog(r + 1) where s and r are the pixel values of the output and the input image and C is a constant

Aim

To apply log transformation to every pixels in the image and observe its transformation

Theory/Discussion

The logarithmic operator is a simple point processor where the mapping function is a logarithmic curve The dynamic range of an image can be compressed by replacing each pixel value with its logarithm. This has the effect that low intensity pixel values are enhanced. Applying a pixel logarithm operator to an image can be useful in applications where the dynamic range may too large to be displayed on a screen (or to be recorded on a film in the first place).

Algorithm

1. Read the image Image
2. Set a value for constant C
3. Apply the log transformation using np.log function to every pixel in the array
4. Save the transformed Image

Code

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal
from PIL import Image
from scipy.misc import imsave
from scipy import fftpack as fftp
import imageio
import skimage.util as skp
import math
```

```
def get_image_horiz():
    img = Image.open('C:/Users/IIST/Downloads/Ragja/6_SC18M003_Ragja_IP2018/horiz.png')
    img.load()
    data = np.asarray( img, dtype="float64" )
    log_transform(data)
def get_image_vertical():
    img = Image.open('C:/Users/IIST/Downloads/Ragja/6_SC18M003_Ragja_IP2018/vertical.png
    img.load()
    data = np.asarray( img, dtype="float64" )
    log_transform(data)
def get_image_cameraman():
    img = Image.open('C:/Users/IIST/Downloads/Ragja/6_SC18M003_Ragja_IP2018/cameraman.ti
    img.load()
    data = np.asarray( img, dtype="float64" )
    log_transform(data)
def log_transform(data):
    C=10
    row,cols=data.shape
    new_data=np.zeros((row,cols))
    for i in range(row):
        for j in range(cols):
            new_data[i,j]=C*math.log(data[i,j]+1)
    print(new_data)
    plt.figure()
    plt.imshow(new_data,cmap='gray')

get_image_horiz()
get_image_vertical()
get_image_cameraman()
```

Result

Log Transformation

Figure 1: Original Cameraman Image

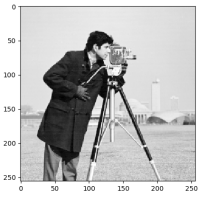


Figure 2: Cameraman Image after Log Transformation

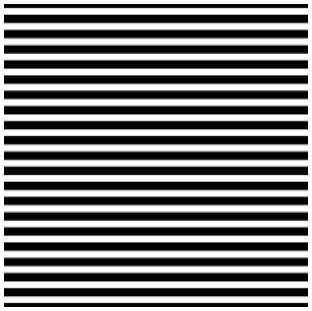


Figure 3: Original Horizontal Image

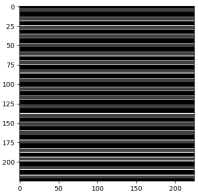


Figure 4: Horizontal Image after Log Transformation

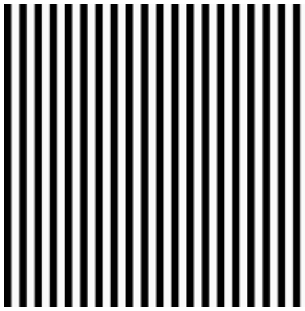


Figure 5: Original Vertical Image

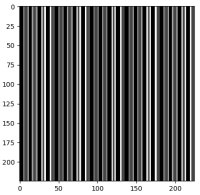


Figure 6: Vertical Image after Log Transformation

Inference

Log Transformation was performed on the given images. Observing the changes in the image after the Log Transformation, we could conclude that a logarithmic transform is appropriate when we want to enhance the low pixel values at the expense of loss of information in the high pixel values.

2. Power Law transformation (Gamma Correction) (use cameraman,horizontal,vertical images)

s = Cr^{γ} where C is the constant and γ is the gamma value (10,8,6,. . .)

Aim

To apply power law transformation to every pixels in the image and observe its transformation

Theory/Discussion

Variation in the value of gamma varies the enhancement of the images. Different display devices / monitors have their own gamma correction, thats why they display their image at different intensity.

This type of transformation is used for enhancing images for different type of display devices. The gamma of different display devices is different.

Algorithm

1. Read the image Image
2. Set a value for constant C
3. Apply the power law transformation from the given formula for different values of gamma
4. Observe differences in the image with change in gamma
5. Save the transformed Image

Code

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal
from PIL import Image
from scipy.misc import imsave
from scipy import fftpack as fftp
import imageio
import skimage.util as skp
import math

def get_image_horiz():
    img = Image.open('C:/Users/IIST/Downloads/Ragja/6_SC18M003_Ragja_IP2018/horiz.png')
    img.load()
    data = np.asarray( img, dtype="float64" )
    power_law_transform(data)
def get_image_vertical():
    img = Image.open('C:/Users/IIST/Downloads/Ragja/6_SC18M003_Ragja_IP2018/vertical.png
    img.load()
    data = np.asarray( img, dtype="float64" )
    power_law_transform(data)
def get_image_cameraman():
    img = Image.open('C:/Users/IIST/Downloads/Ragja/6_SC18M003_Ragja_IP2018/cameraman.ti
    img.load()
    data = np.asarray( img, dtype="float64" )
    power_law_transform(data)
def power_law_transform(data):
    C=10
    row,cols=data.shape
    new_data=np.zeros((row,cols))
```

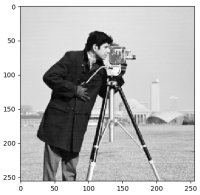
```
    for i in range(row):
        for j in range(cols):
            new_data[i,j]=C*math.pow(data[i,j],1/8)
    print(new_data)
    plt.figure()
    plt.imshow(new_data,cmap='gray')

get_image_horiz()
get_image_vertical()
get_image_cameraman()
```

Result

Log Transformation

Figure 7: Original Cameraman Image



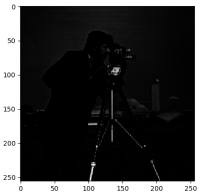


Figure 8: Cameraman Image after Power Law Transformation - Gamma 1/8 and 8

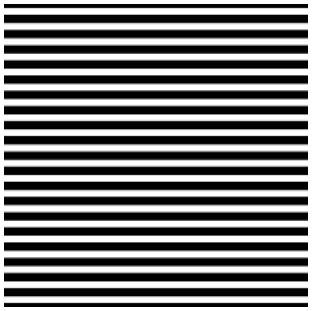


Figure 9: Original Horizontal Image

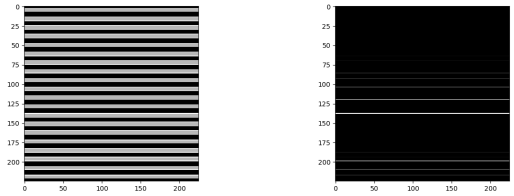


Figure 10: Horizontal Image after Power Law Transformation - Gamma 1/8 and 8

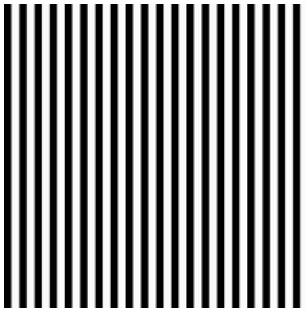


Figure 11: Original Vertical Image

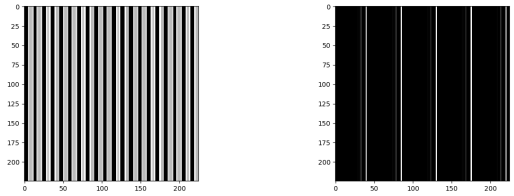


Figure 12: Vertical Image after Power Law Transformation - Gamma 1/8 and 8

Inference

Power Law Transformation was performed on the given images. Gamma values were varied and the changes were observed in the images.

3. Perform histogram equalization of an image.

Compare the histograms of input and output images. Comment on what happens if we equalize an already histogram equalized image.

Aim

To learn how to perform histogram equalization of an image

Theory/Discussion

Histogram equalization is a method to process images in order to adjust the contrast of an image by modifying the intensity distribution of the histogram. The objective of this technique is to give a linear trend to the cumulative probability function associated to the image

Algorithm

1. Read the image Image
2. Flatten the image array
3. Find the CDF of the image using plot.hist()
4. Equalize the CDF by multiplying every value with the grey level - 255
5. Obtain image back by reshaping the array

Code

```
from matplotlib import pyplot as plt
import numpy as np
def get_image():
    img = plt.imread('C:/Users/IIST/Downloads/Ragja/6_SC18M003_
    Ragja_IP2018/cameraman.tif')
    data = np.asarray( img, dtype="float64" )
    img=img.flatten()
    plt.figure()
    cdf, bins, patch=plt.hist(img, bins=64, range=(0,256),
    normed=True, cumulative=True)
    new_pixels = np.interp(img, bins[:-1], cdf*255)
    new_image = new_pixels.reshape(data.shape)
    plt.imshow(new_image, cmap='gray')
    #cdf, bins, patche=plt2.hist(new_pixels, bins=256,
    range=(0,256), normed=True, color='blue', alpha=0.4, cumulative=True)

get_image()
```

Result

Histogram Equalization

Figure 13: Original Cameraman Image

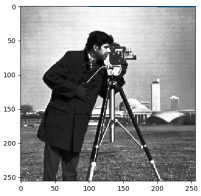


Figure 14: Cameraman Image after Equalization

Inference

Histogram Equalization was performed on the cameraman image. The output does not change if an already equalized image is tried to be equalized again

4. Histogram Specification

Implement a program to match the histogram of the output image for 0.png to the histogram provided in histG.mat.
Compare and comment on the histograms of the input image, equalized image, histogram given for matching and the histogram of the matched image.
Steps:
(a) Equalize histogram of input image.
(b) Equalize histogram of specified image (If the specified histogram is already equalized,skip this step).
(c) Relate the two equalized histograms.

Aim

To learn how to perform histogram specification/matching between two images

Theory/Discussion

Histogram matching or histogram specification is the transformation of an image so that its histogram matches a specified histogram. Procedure: We first equalize the histogram p_x of the input image x:

$$y = f(x) = \int_0^x p_x(u)du$$

We then equalize the desired histogram p_z of the output image z:

$$y' = g(z) = \int_0^z p_z(u)du$$

The inverse of the above transform is

$$z = g^{-1}(y')$$

As the two intermediate images y and y' both have the same equalized histogram, they are actually the same image, i.e., $y = y'$, and the overall transform from the given image x to the desired image z can be found to be:

$$z = g^{-1}(y') = g^{-1}(y) = g^{-1}(f(x))$$

However, as the image gray levels are discrete, the continuous mapping obtained above can only be approximated. The discrete histograms $h_y[i]$ and $h_{y'}[i]$ are not necessarily identical. We therefore need to relate each gray level in x with $h_x[i] \neq 0$ to a gray level in z with $h_z[j] \neq 0$, so that the mapping from y to y' can be established.

Algorithm

1. **Step 1:** Find histogram of input image h_x, and find its cumulative H_x, the histogram equalization mapping function:
$$H_x[j] = \sum_{i=0}^{j} h_x[i]$$
2. **Step 2:** Specify the desired histogram h_z, and find its cumulative H_z, the histogram equalization mapping function:
$$H_z[j] = \sum_{i=0}^{j} h_z[i]$$
3. **Step 3:** Relate the two mapping above to build a lookup table for the over all mapping. Specifically, for each input level i, find an output level j so that $H_z[j]$ best matches $H_x[i]$:
$$|H_x[i] - H_z[j]| = min_k|H_x[i] - H_z[k]|$$
and then we setup a lookup entry $lookup[i] = j$.

Code

```
import scipy.io
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import pyplot as plt2
from PIL import Image

def find_nearest(array, value):
    array = np.asarray(array)
    idx = (np.abs(array - value)).argmin()
    return idx

def get_histograms_ready():
    mat = scipy.io.loadmat('histG1_1.mat')
    hist_array=mat['histG']
    hist_array=np.array(hist_array)
    sum=0
    for i in range(len(hist_array)):
        sum=sum+hist_array[i]
    hist_array=hist_array/sum
    cdf_array=np.zeros((len(hist_array)))
    for i in range(len(hist_array)):
        if i==0:
            cdf_array[i]=hist_array[i]
        else:
            cdf_array[i]=cdf_array[i-1]+hist_array[i]
    cdf_array=255*cdf_array
    #plt.plot(cdf_array)
    img2 = Image.open('C:/Users/IIST/Downloads/Ragja/6_SC18M003_Ragja_IP2018/0.png').co
    data=np.array(img2)

    historg=np.array(img2.histogram())/(data.shape[0]*data.shape[1])
    cdf2_array=np.zeros((len(historg)))
    for i in range(len(historg)):
        if i==0:
            cdf2_array[i]=historg[i]
```

```
            else:
                cdf2_array[i]=cdf2_array[i-1]+historg[i]

        cdf2_array=255*cdf2_array
        #plt.plot(cdf2_array)
        hist_specification(hist_array,cdf_array,cdf2_array,data)


def hist_specification(pdf1,cdf1,cdf2,data):
    look_up=np.zeros((cdf1.shape[0]))

    for i in range(cdf1.shape[0]):
        look_up[i]=find_nearest(cdf2,cdf1[i])

    imgeq=np.zeros((data.shape[0],data.shape[1]))
    for i in range(0,data.shape[0]):
        for j in range(0, data.shape[1]):
            imgeq[i,j] = look_up[int(data[i,j])]

    imgfinal = Image.fromarray((imgeq).astype('uint8'))
    imgfinal.save('equalized_image.png')
    plt.subplot(313),plt.hist(imgeq),plt.title('after histogram equalization')
get_histograms_ready()
```

Result

Histogram Specification

Figure 15: Original 0 Image

Figure 16: 0 Image after Histogram Matching

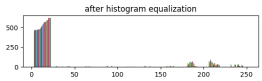


Figure 17: Plot after Histogram Specification

Inference

Histogram Specification was performed for the given image and histogram

5. Filtering

Perform high pass and low pass filtering in a noisy image and comment on your observations.
Steps:
(a) Select a noisy image (with any noise).
(b) Take the fft of the image.
(c) Select two suitable kernels (one for low pass filtering and other one for high pass filtering).
(d) Zero pad it to make it to the size of image.
(e) Multiply it with the fft of your noisy image.
(f) Take the inverse fft.
Display the noise,noisy image,filter used,multiplied fft and inverse fft.

Aim

To learn how to add/denoise impulse and gaussian noises on an image in frequency domain

Theory/Discussion

Images taken with both digital cameras and conventional film cameras will pick up noise from a variety of sources. To use these images further, the noise should be (partially) removed for aesthetic purposes as in artistic work or marketing, or for practical purposes such as computer vision. From the Convolution Theorem, convolution in spatial domain is same as multiplication domain. Hence the image and filter are transformed to the frequency domain and then finally inverse FFT is performed to obtain the filtered image

Algorithm

1. Read the image Image
2. Add noise to the Image
3. Decide on the LPF and HPF kernels.
4. Pad the kernels to the size of the image
5. Perform FFT on the image and kernels
6. Multiply the FFTed kernel and image
7. Take inverse FFT to obtain the image back

Code

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal
from PIL import Image
from scipy.misc import imsave
from scipy import fftpack as fftp
import imageio
import skimage.util as skp

hpf=np.matrix([[-1,-1,-1],[-1,8,-1,],[-1,-1,-1]])
lpf=np.matrix([[1,1,1],[1,1,1],[1,1,1]])
lpf = np.float_(lpf)
lpf=lpf/9
hpf = np.float_(hpf)
hpf=hpf/9

final_filter = np.zeros((256,256))
final_filter[:hpf.shape[0],:hpf.shape[1]] = hpf
print(final_filter)
f_filter=np.fft.fft2(final_filter)

final_filter2 = np.zeros((256,256))
final_filter2[:lpf.shape[0],:lpf.shape[1]] = lpf
print(final_filter2)
f_filter2=np.fft.fft2(final_filter2)

img = Image.open('C:/Users/IIST/Downloads/Ragja/6_SC18M003_Ragja_IP2018/gaussian_noise.p
img.load()
data = np.asarray( img, dtype="float64" )

noise=skp.random_noise(data,mode='gaussian')
imsave('gaussian_noise_only.png',noise)

image_fft=fftp.fft2(data)

final_image=image_fft*f_filter
image=np.fft.ifft2(final_image)
img_shft = np.fft.fftshift(final_image)
amp_fft=20*np.log(np.abs(img_shft))
plt.figure()
plt.imshow(amp_fft, cmap='gray')
```

```
final_image2=image_fft*f_filter2
image2=np.fft.ifft2(final_image2)
img_shft = np.fft.fftshift(final_image2)
amp_fft=20*np.log(np.abs(img_shft))
plt.figure()
plt.imshow(amp_fft, cmap='gray')

img=imageio.imwrite('5_inverse_lpf.png',image2)

img=imageio.imwrite('5_inverse_hpf.png',image)
```

Result

Denoising Gaussian Noise

Figure 18: Original Cameraman Image

Figure 19: Cameraman Image with Gaussian Noise

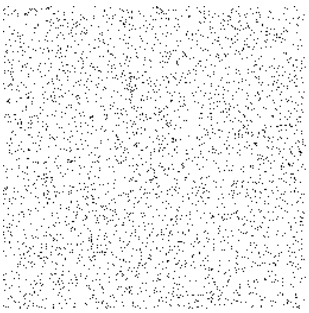


Figure 20: Noise in the Image

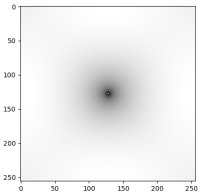


Figure 21: High Pass Filter FFT

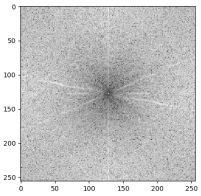


Figure 22: Multiplied FFT and HPF kernel FFT

Figure 23: Inverse FFT after HPF

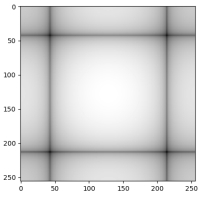


Figure 24: Low Pass Filter FFT

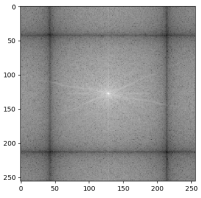


Figure 25: Multiplied FFT and LPF kernel FFT

Figure 26: Inverse FFT after LPF

Inference

Noise was added to the image. Subsequently denoising was performed by low pass and high pass filter in the frequency domiain. It was found that low pass filter worked well for gaussian noise(low pass filtering to remove white noise)