Image Processing:
Gray Level Transformation

Author Name: Dr. Rubina Shahin Zuberi
I.T.S Engineering College, Greater Noida

Prerequisites: Digital Image Processing basics, knowledge of functions like logarithmic, exponential and piece-wise linear functions.

Software required: OpenCV 2.4.13.6, Python 2.7.15, NumPy 1.14.5

Lecture Notes:

1. Introduction:

What is Gray-level transformation in Image Processing?

The Gray-level transformation is one of the image enhancement technique. Image enhancement techniques are devised to provide better images for human perception and also for providing inputs to autonomous systems or machines.

The enhancement methods can be divided into two broad categories: Spatial domain and Frequency/transform domain.

Spatial domain & Frequency/ Transform domain

Spatial domain: If the image processing algorithm operates directly on the input image to change the intensities at the pixel level, it is said to be working in the spatial domain. Two principal categories of spatial processing are intensity or gray-level transformation and spatial filtering.

Gray-level transformation operates on every pixel of the input image primarily for contrast manipulation and image thresholding as shown in Figure 1 & 2. Spatial filtering generally works for the image sharpening in the neighbourhood of every pixel in the image as shown in Figure 3.

(a) Input image (b) Output image

Figure 1: Image contrast manipulation using Gray-level transformation

(a) Input image (b) Output image

Figure 2: Image thresholding using Gray-level transformation

(a) Input image (b) Output image

Figure 3: Image sharpening using spatial filtering

Frequency/Transform Domain: In transform domain, the input image is first transformed using Fourier transform and then processed in the transform domain. The output image is obtained using inverse transform so as to bring the results back into the spatial domain. In this lesson, we will focus only on the spatial domain.

- The comparison between the above two domains in image processing is not possible as some image processing tasks are more meaningful to implement in the spatial domain while for some the transform domain proves to be the best.
- The spatial domain techniques, like gray-level transformation are computationally more efficient than transform domain techniques and these require less processing resources to implement.

2. Background for Gray-level Transformation:

The spatial domain processes like gray-level transformation can be denoted by the following expression:-

$$g(x, y) = T[f(x, y)]$$

here f(x, y) is the input image,

g(x, y) is the output image,

T is an operator# on f (defined over a neighbourhood of point (x, y) as shown in Figure 4)

The operator can apply to a set of images as well for e.g. performing the pixel by pixel sum of a sequence of images for noise reduction.

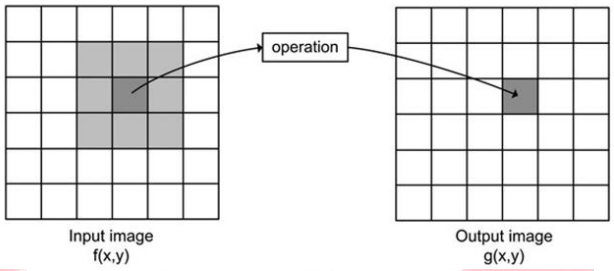


Figure 4: Neighbourhood about the point being operated upon by T

The origin (central dark gray cell in the input image of Figure 4) of the neighborhood is being moved from pixel to pixel and operator is applied to the pixels in neighbourhood to yield the output at that location.

In gray-level transforms, the image is processed on its pixel level. The intensity (or the gray-level) of the pixel of an image is being modified using gray-level transformation functions like logarithmic function, power-law or gamma transformation, Fourier transform etc.

The smallest possible neighbourhood has size 1X1 where g depends only on the value of f at say (x, y) and T. This means gray-level or intensity transformation function operates on a single point:

$$s = T(r)$$

Here s and r denotes variables respectively of g and f at the point (x, y) such that T becomes the gray-level or intensity transformation function. The examples of T includes contrast stretching and thresholding.

3. Gray level Transformation:

For getting started with the computations involved in gray-level transformations on an image, gray scale model and the three basic frequently used mathematical functions are of great help.

Gray Scale Model:

In a gray scale image, each pixel of the image corresponds to the intensity level (amount of light which it reflects). These images are popular as black and white and the values of the intensities varies from dullest black to brightest white containing numerous shades of gray as shown in Figure 5. They are different from one bit bi-level binary images which have got only two colors pure black and white.

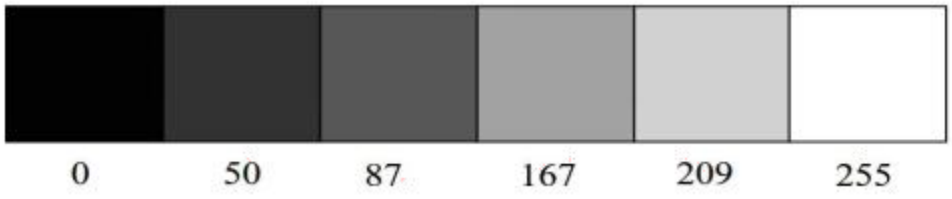


Figure 5: Gray Scale Levels

The intensity of the pixels ranges from 0 (0%) to 1 (100%). Here 0 is for black colour and 1 is for absolute white while in between gray shades can be represented by fractions between 0 and 1. The white colour is depicted using 255, while black colour is depicted using 0.

Gray codes can be converted to colour codes and vice versa. (Refer color models)

Basic transformation functions:

Three basic types of functions used frequently for image enhancement are: linear, logarithmic and power-law or gamma transformation functions as shown in Figure 6.

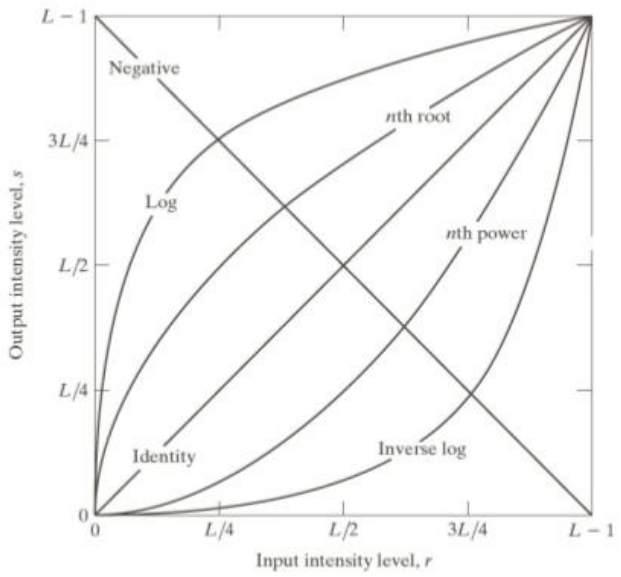


Figure 6: Output vs input intensity comparisons

Linear Transformation for negatives of images:

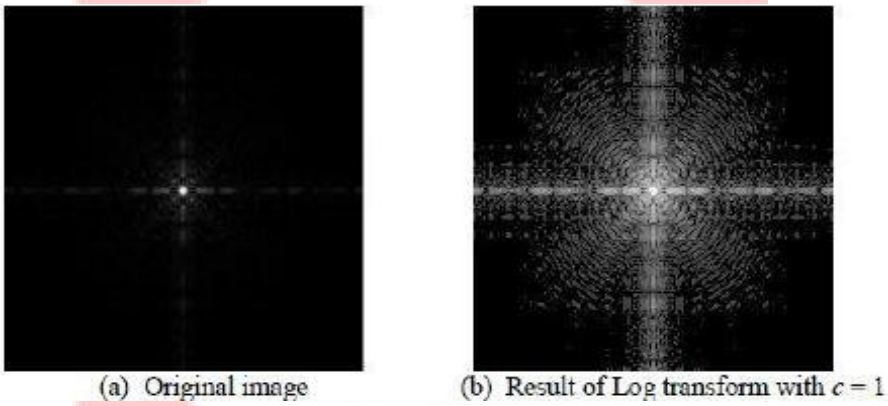
The negative of an input image with intensities varying in [0, L-1] is obtained by using the negative transformation,

$$s = L-1-r$$

This may be suited for enhancing white or gray detail embedded in the dominant dark areas of an image.

Log Transformations:

The log function compresses the dynamic range of images with large amount of variations in pixel values. The log curve in Figure 7 shows that the transformation maps a narrow range of low intensity values of the input image into a wider variety of levels in the output image.



(a) Original image (b) Result of Log transform with $c = 1$

Figure 7: Effect of logarithmic transform to visualize the diffraction pattern of an image.

The general form of the log transformation is:-

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

here c is a constant and r>=0.

Gamma(γ) or Power-law transformations:

The devices used for display, printing, image capture like Cathode Ray Tube devices, scanners, printers use gamma correction before producing an output image.

Power-law transformations can be denoted by,

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

here c and γ are positive constants. Figure 8 shows power-law curves with fractional values of γ mapping narrow range of dark input values into a wider range of output values. For higher values of input levels the opposite is true. The process required to adjust the power-law response phenomenon is called gamma correction. Moreover, power-law transformations are useful for general purpose contrast manipulation.

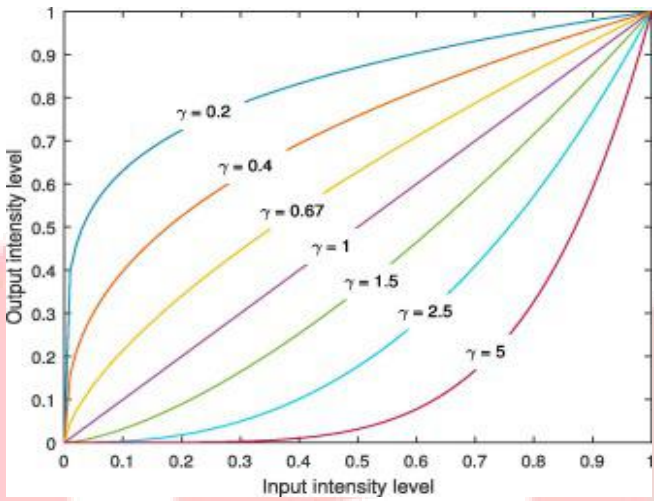


Figure 8: Plots showing power-law for various values of γ (c=1 for all cases)

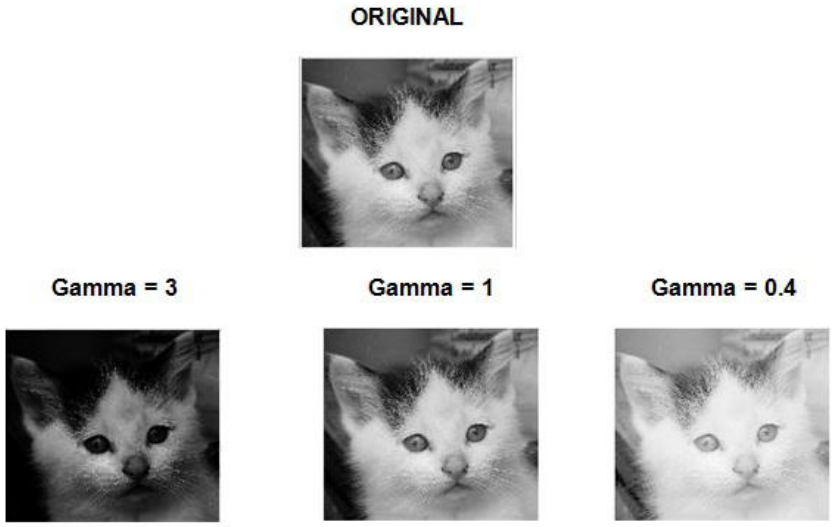


Figure 9: Effect of changing γ values in Gamma or Power-law transformations

Piecewise-Linear transformations:

Piecewise linear functions can be arbitrarily complex and hence can represent more complex transformations, their specification however requires considerably more user input. The piecewise linear functions includes- contrast stretching, gray level slicing, bit-plane slicing etc.

Contrast stretching: One of the simplest piecewise linear functions is contrast stretching. It may be used for low contrast images which results due to low range of sensor, poor illumination etc.

It is a process that expands the range of intensity levels in an image so that it spans the full intensity range of the recording medium or display device.

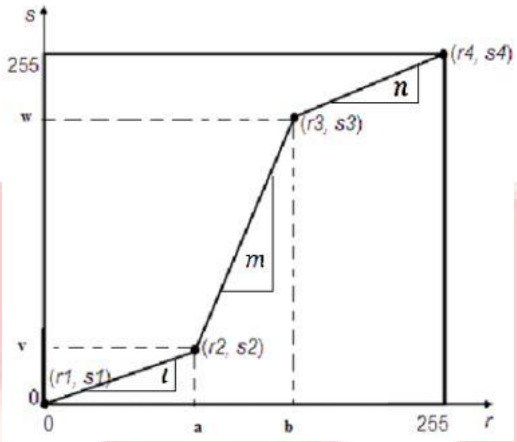
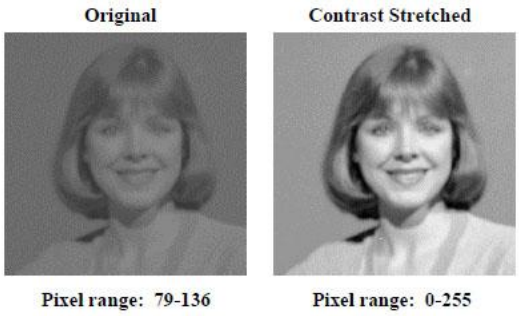


Figure 10: Contrast stretching.

The x-axis "r' depicts Input Intensity levels and y-axis 's' depicts Output Intensity levels as shown in Figure 10. It shows stretching of r1 is done using slope 't', r2 using slope 'm' and r3 using slope 'n'. Resulting in stretched points at s1, s2 and s3. The effect of such transformations are of the form depicted in Figure 11.



(a) Input Image (b) Output Image

Figure 11: Contrast stretching

Gray-level slicing: Some applications may need the enhanced intensity levels only in certain gray level values. In such cases, gray level slicing is used.

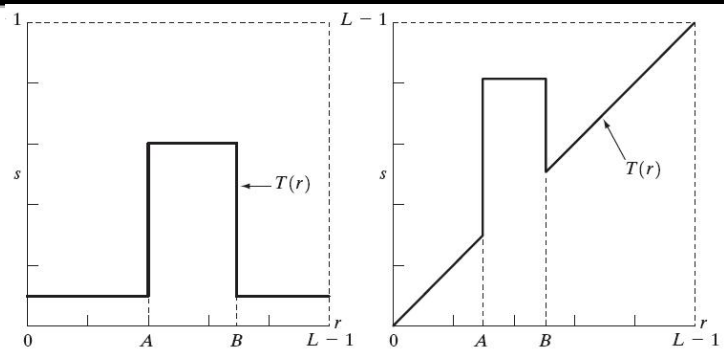


Figure 12: Gray-level slicing

As shown in Figure 12 in both the images, the gray level is highlighted in the range [A-B]. But, the gray levels are reduced in the left image while are preserved in the right image.

Bit-plane slicing: It is used to highlight the contribution made to total image appearance by specific bits.

4. OpenCV functions (to be used in the example programs):

I. Functions for Simple Thresholding:-

Simple thresholding is a special case of contrast stretching which is a piecewise linear function. It is sometimes called image binarization as well. As it results in a binary gray image which contains only two levels. It sometimes proves to be a very simple and useful technique of contrast enhancement.

cv2.threshold:

- 1st argument is the source image, which **should be a grayscale image**.
- 2nd argument is the threshold value which is used to classify the pixel values.
- 3rd argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value.
- 4th argument provides different styles of thresholding.

  Different types are:

  cv2.THRESH_BINARY

  cv2.THRESH_BINARY_INV

  cv2.THRESH_TRUNC

  cv2.THRESH_TOZERO

  cv2.THRESH_TOZERO_INV

Two outputs are obtained. First one is a **retval** and second output is **thresholded image**.

5. Example Programs:

I. Negative of Image:-

Problem Statement: Take a RGB image. First convert it to grayscale and then display the negative of that image.

Program Snippet:

```
#import OpenCV and NumPy
import cv2
import numpy as np

#read an image
original_img = cv2.imread('test.jpg')

#convert an image to grayscale
img = cv2.imread('test.jpg',  cv2.IMREAD_GRAYSCALE)
print ("r=",img)

#perform linear transformation and negate the image
s = 255 - img
print ("\n\ns=",s)

#display the images
cv2.imshow('Original Image', original_img)
cv2.imshow('Grayscale Image', img)
cv2.imshow('image negative', s)
cv2.waitKey(0) #Display images until a key is pressed
```

Output:

```
>>> ================================ RESTART ================================
>>>
r= [[255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 ...
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]]


s= [[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

Original Image	Grayscale Image	Image Negative
		

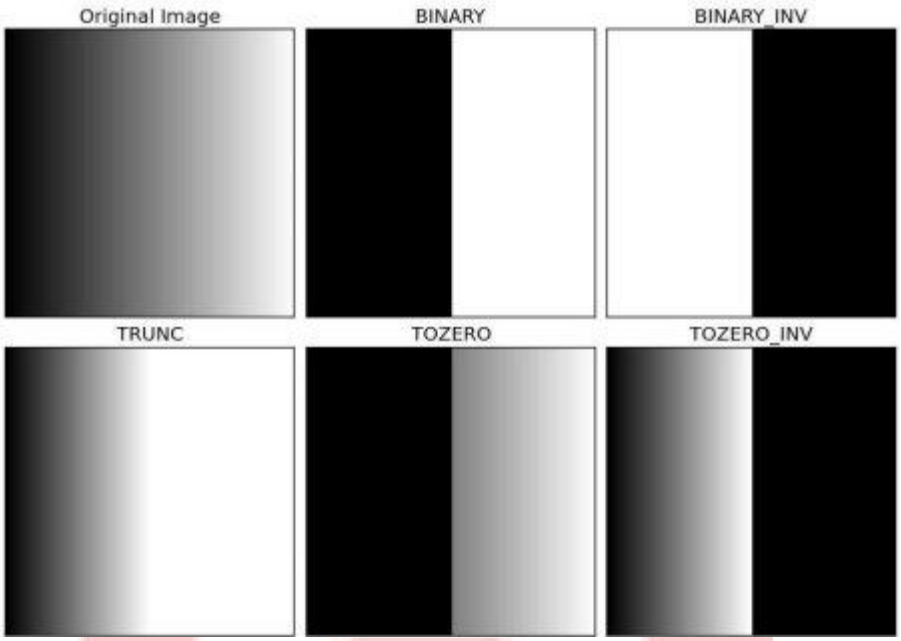
II. Simple Thresholding:-

Problem Statement: If pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black). The function used is **cv2.threshold**.

Program Snippet:

```
#import OpenCV and NumPy
import cv2
import numpy as np


#import matlabplot into NumPy plot
from matplotlib import pyplot as plt

# read an image
img=cv2.imread('gradient.png',0)

# simple binary thresholding is done from level 127 to 255 the return value is saved in
# thresh1
ret,thresh1=cv2.threshold(img,127,255,cv2.THRESH_BINARY)

# inverted binary thresholding is done from level 255 to 127 the return value is saved in
# thresh2
ret,thresh2=cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)

# truncated binary thresholding is done from truncation level  to 255 the return value is
# saved in thresh3
ret,thresh3=cv2.threshold(img,127,255,cv2.THRESH_TRUNC)

# setting threshold value to zero(minimum) saving in thresh4
ret,thresh4=cv2.threshold(img,127,255,cv2.THRESH_TOZERO)

# setting threshold value to maximum (hence zero inversion) saving in thresh5
ret,thresh5=cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)

# giving titles to img thresh1, 2, 3, 4 and 5
titles=['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images=[img,thresh1,thresh2,thresh3,thresh4,thresh5]

# using for loop plot thresholded images from 1 to 5 as two dimensional x-y plot
for iinxrange(6):
plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
plt.title(titles[i])
plt.xticks([]),plt.yticks([])

# display the plots
plt.show()
```

Output:



References:

[1]. http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html
[2]. Digital Image Processing book by Rafael C. Gonzalez and Richard Eugene Woods