3 Chapter Three

Intensity Transformation and Spatial Filtering

This chapter will deal with different image processing techniques when an image is present in spatial domain. General representation of spatial domain image transformation is given as:

$$g(x,y) = T[f(x,y)] \quad \mathbf{(3-1)}$$

Here are few basic concepts that are associated with spatial domain processing.

- $Intensity\ thresholding$: For the simplest case of binary thresholding, we decide that pixel values of lower than some specified threshold be changed to zero and higher values are changed to L-1 where L is the number of intensity levels.
- $Contrast\ stretching$: If k is intensity threshold, the process that darkens the intensity levels that are below k and brightens the pixels which are above threshold is called the contrast stretching
- $Image\ enhancement$: It is the process of manipulating the image pixel values such that the information in resulting image is clearer than the original.

3.1 Basic Image Transformations

There are three types of basic image transformation:

- Linear Transformation
- Non-Linear Transformation
  - Logarithmic Transformation
  - Power Law Transformation

3.1.1 Linear Transformation

In this transformation, the intensity value of a pixel is changed using some linear operation. This change may be increase or decrease in intensity value of an image. For example, to enhance the visibility of an image, intensity value of each pixel can be scaled up to some defined level. Linearly upscaled image shown in figure 3.1-1 have better visual information as compared to original image shown in figure 3.1-1 A. Similarly, linear intensity scaling can enhance the quality and color in color images.

$$s = r \pm i \quad \mathbf{(3-2)}$$

Negative of an image is obtained by subtracting each pixel intensity value from maximum intensity value that can be in the image. In other words, each intensity value represents the intensity level. If there are L distinct intensity levels in an image and r is the input pixel intensity value then output pixel intensity value can be calculated as:

$$s = L - 1 - r \quad \mathbf{(3-3)}$$

In normal gray level image, the intensity values range from $0 - 255$ $(0 - L - 1)$. If a pixel has intensity value 222 then its negative will be 255-222=33.

Python is very versatile and feature rich language. It supports both Single Instruction Single Data (SISD) and single instruction multiple data (SIMD) operations. SIMD operations are also called $vector\ operation$. For example, we can replace lines 8-12 in program P3-01 with single statement.

```
Img2=(L-1)-img1
```

This single subtraction operation will be applied on all pixels of `img1` and result will be assigned to `img2` on corresponding positions. There are some useful applications of image negative. In gray scale images, image negatives are used to enhance the gray level details which is present in darker areas. Specially if dark areas are dominant in image, information can't be seen directly. Taking image negative can enhance the contrast of image darker regions resulting in better visualization. For example, in image containing blood vessels, vessels can better be seen in negative image. Similarly, color image negative can provide information which is not clear in standard color space.

3.1.2 Log Transformation

In this transformation, log function is applied on intensity value of each pixel of input image. Equation 3-3 shows the process.

$$s = c * log(1 + r) \qquad \mathbf{(3-4)}$$

Log of large values results in small values. For example, if $K = [1,2,4,8,16,32,64,128]$, taking its $\log_2 K$ produces values that lie in shorter range i.e. $[0, 1, 2, 3, 4, 5, 6, 7]$. It can be observed that log transformation has compressed the intensity values. The log function has the important characteristic that it compresses the dynamic range of pixel values. The term *dynamic range* refers to the ratio of max and min intensity values. There are the imaging modalities that can create images with pixel intensity values beyond the standard range of 0-255. In that case, log transformation can play its role to bring larger values in standard range. There are two major application areas of log transformation:

- Expanding the dark pixels and compressing the corresponding bright pixels
- Compressing the dynamic range

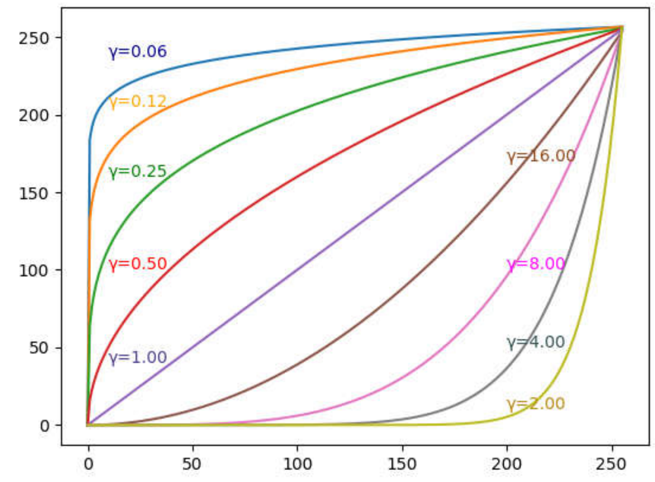
3.1.3 Power law (Gamma) Transformation

Power law transformation is given by

$$s = cr^{\gamma} \qquad where \qquad c, \gamma \geq 0 \quad \mathbf{(3-5)}$$

Where c and γ are some positive constants. If we consider c=1, we can easily describe the behavior of γ constant. It is a non-linear operation and like log transformation, gamma transformation also compresses and expands the intensity values. A gamma value $\gamma < 1$ is sometimes called an encoding gamma, and the process of encoding with this compressive power-law nonlinearity is called ***gamma compression***; conversely a gamma value $\gamma > 1$ is called a decoding gamma and the application of the expansive power-law nonlinearity is called ***gamma expansion***. Following graph represents the compression behavior for different values of gamma parameter.

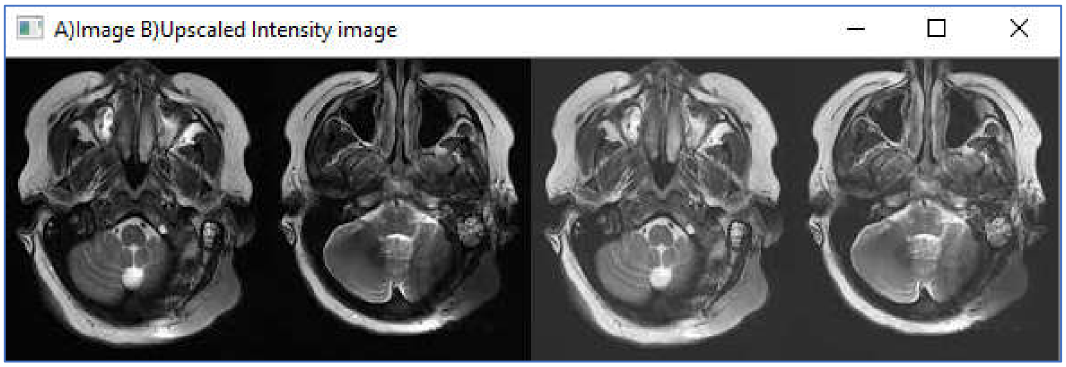
As the value of γ goes lower, low intnsity levels are compressed more and high intensity levels are stretched more. In case of γ value going higher than one, low intensity values are spread more and high intensity values are compressed more. When applying gamma transformation, one should be very careful about choosing the values of parameters. It would be good exercise, if you do some experimentation with the code used to generate this graph.



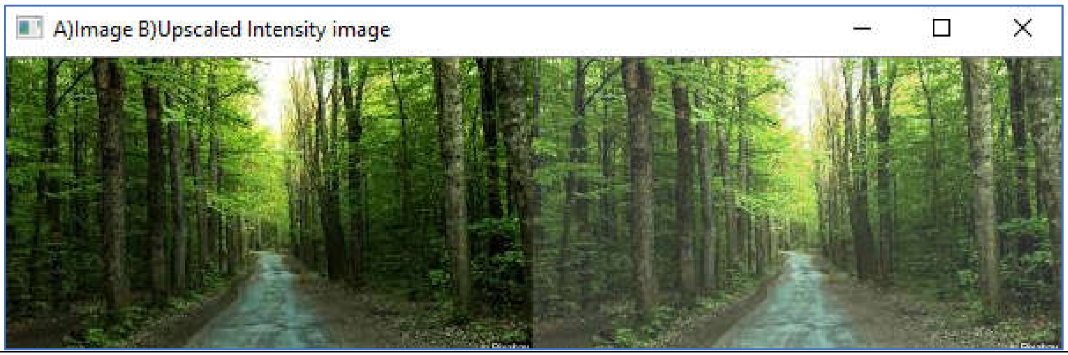
When working with transformations in OpenCV, you must be careful about image data types and their intensity ranges. Considering the intensity resolution, there are two modes to work with images 1) operate on raw intensity values and 2) work with normalized intensity values. Operations till now, we have seen, are based on raw intensity values where pixel intensity ranges from 0-255. However, some of the operations require to normalize the pixel intensity value in the range 0-1 like gamma transformation. This can be easily done by dividing image each pixel with the maximum intensity value found in that image i.e.

$$norm(img) = \frac{img(i)}{max(img)} \qquad \forall\, i\ \in L-1 \qquad \mathbf{(3-6)}$$

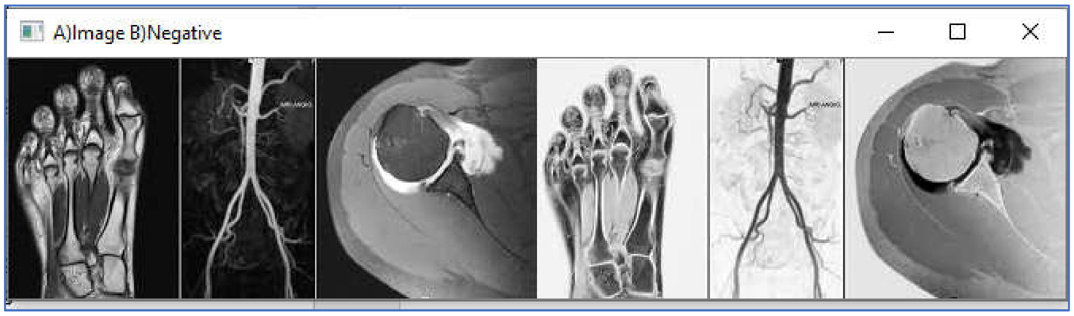
A)



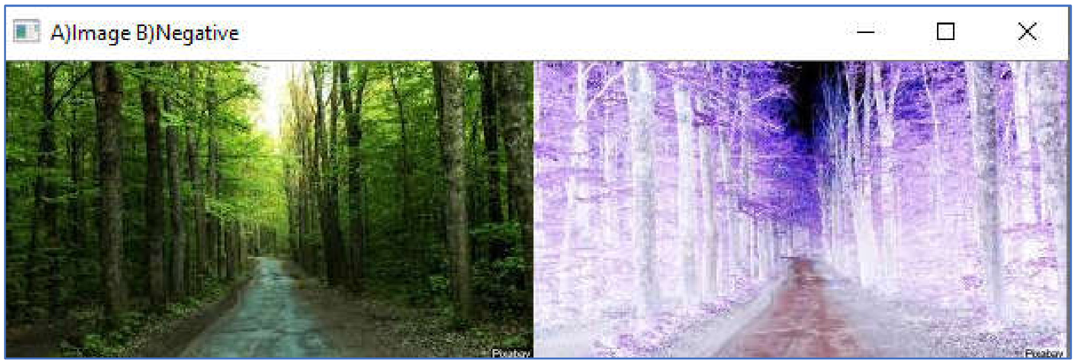
B)



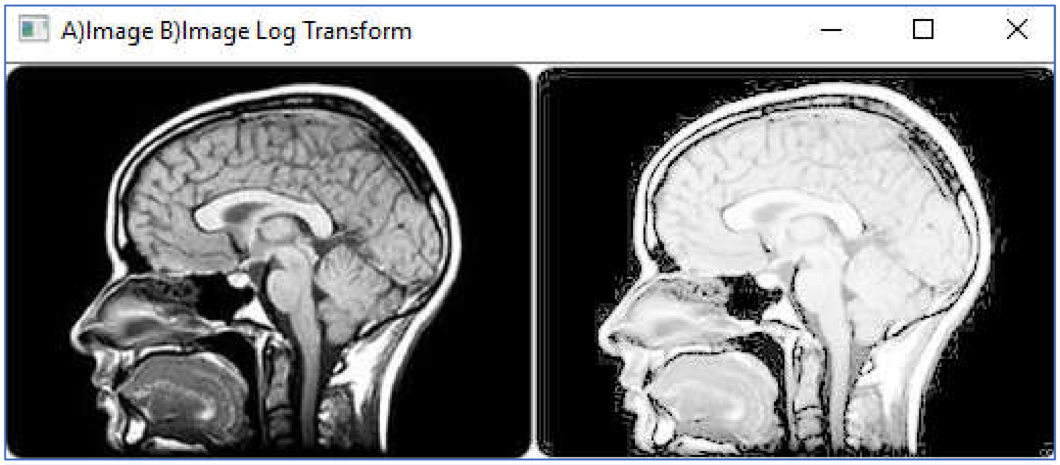
C)


A)Image B)Negative

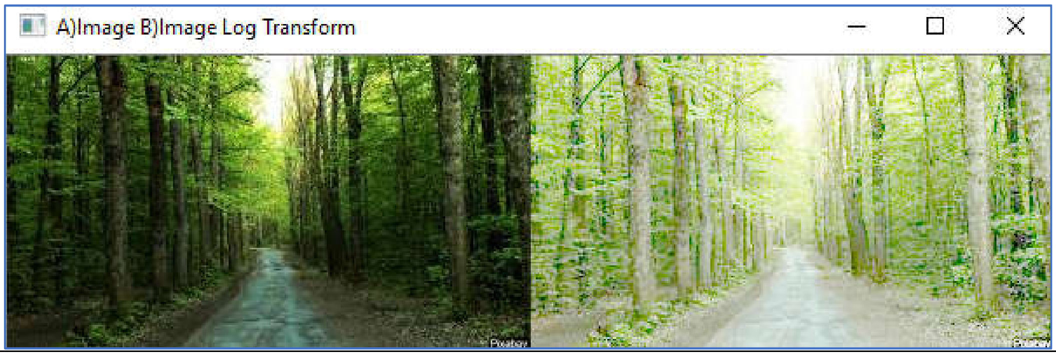
D)


A)Image B)Negative

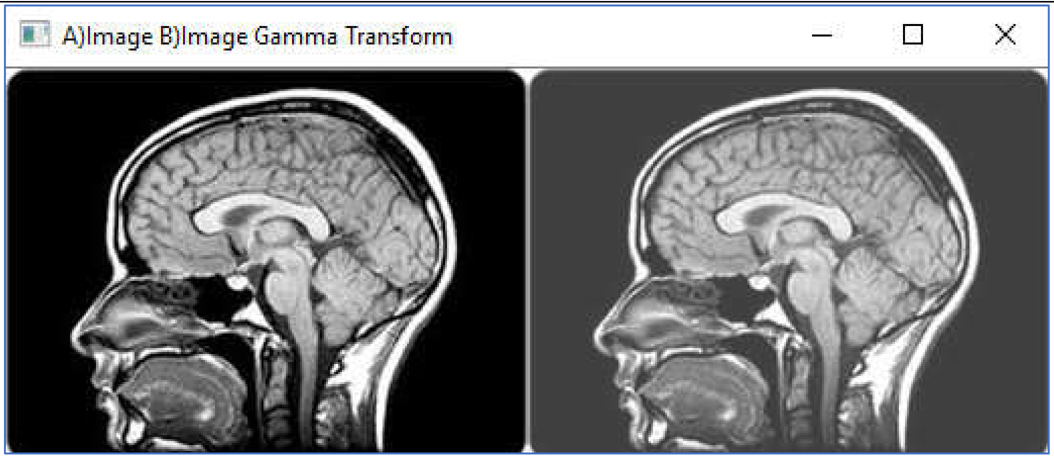
E)


A)Image B)Image Log Transform

F)


A)Image B)Image Log Transform

G)



H)

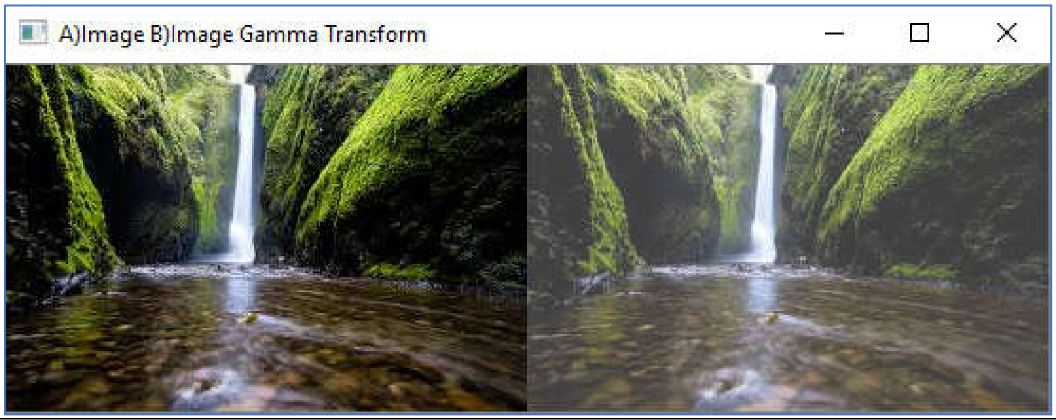


Figure 3-1: A-B) Linear Image Transformation in Grayscale and Color Images. C-D) Grayscale and Color Image Negatives E-F) Log Transformation on Grayscale and Color Images G-H) Gamma Transformation on Grayscale and color images.

Reference Programs

***Program P3-01**: Grayscale linear intensity transformation*

***Program P3-02**: Color linear intensity transformation*

***Program P3-03**: Grayscale image negative*

***Program P3-04**: Color image negative*

***Program P3-05**: Grayscale image log transformation*

***Program P3-06**: Color image log transformation*

***Program P3-07**: Drawing graph of gamma transformation*

***Program P3-08**: Grayscale image gamma transformation*

***Program P3-09**: Color image gamma transformation*

3.2 Piecewise Linear Transformation

Transformations discussed in last section operate on whole image. There may be the situations where we need to process different parts of an image differently by applying different type of transformations on difference pieces of image.

3.2.1 Piecewise Linear Thresholding (PLT)

An image contains pixel that have intensity range $0 \rightarrow L-1$. Simplest of such transform is to define a threshold or boundary T that divide all intensity levels to two parts i.e. $0 \le p_1 \le T$ *and* $T+1 \le p_2 \le L-1$. For example, we may decide T=127 and change all intensity levels below this threshold to zero and all intensity levels above this threshold to 255.

$$s = T(r) = \begin{cases} r = 0 & if\ r \le T \\ r = 255 & if\ r \ge T+1 \end{cases} \qquad (3-7)$$

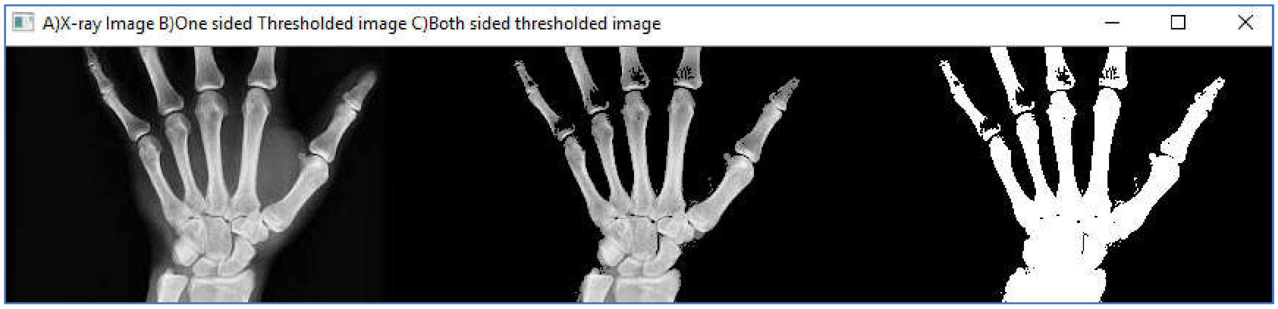

A)X-ray Image B)One sided Thresholded image C)Both sided thresholded image

Figure Error! Use the Home tab to apply 0 to the text that you want to appear here.**-1: Application of multi-level intensity thresholding – T=105**

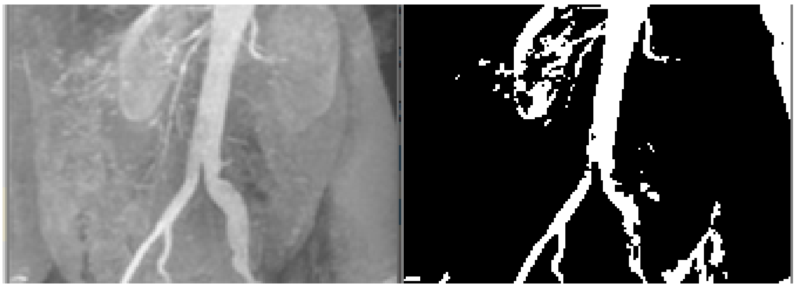
3.2.2 Intensity Level Slicing

There are applications in which it is of interest to highlight a specific range of intensities in an image. Some of these applications include enhancing features in satellite imagery, such as masses of water, and enhancing flaws in X-ray images. If we decide two thresholds T_1 and T_2 where

- Intensity values below T_1 are transformed using some function $T_1(r)$,

- Intensity values from $T_1 + 1$ to T_2 are transformed using function $T_2(r)$, and
- Intensity values above T_2, are transformed using function $T_3(r)$.

The values of T_1 and T_2 can be either found manually or using some sophisticated algorithm. This process is known as multi-thresholding.



3.2.3 Contrast Stretching

Contrast can be defined as the number of distinct colors in an image. In a grayscale image that contain intensity values from 0-255, it can be said that such image contains 256 colors ($L_{max} = 255$). Contrast can also be defined as difference between highest and lowest intensity values. If L is intensity value then

$$contrast = L_{highest} - L_{lowest} \quad \mathbf{(3-8)}$$

A grayscale image may contain contrast value below L_{max}, this allows us to use all gray colors in image by expanding the intensity distribution known as **contrast stretching**. Here is simple formula for contrast stretching:

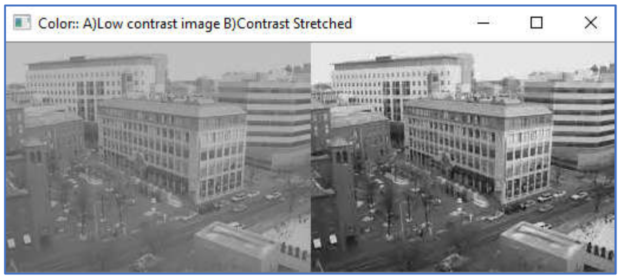
$$s_i = (r_i - r_{min}) * \frac{L_{max} - L_{min}}{r_{max} - r_{min}} + L_{min} \quad \mathbf{(3-9)}$$

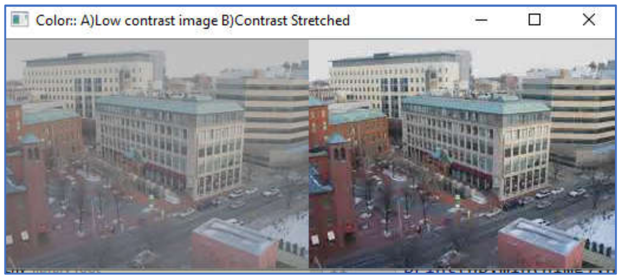
Where

$s_{i,}\, r_i = Intensity\ value\ of\ ith\ output\ and\ input\ pixel\ respectivelly$

$r_{min}, r_{max} = Lowest\ and\ highest\ intensity\ value\ in\ input\ image$

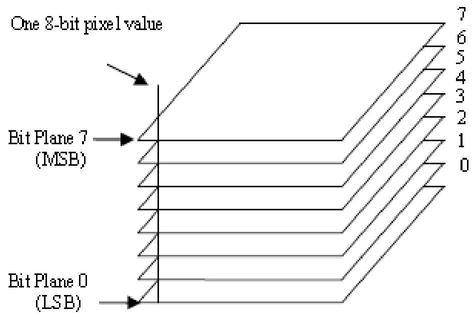
$L_{min}, L_{max} = Lowest\ and\ highest\ intensity\ value\ of\ target\ contrast\ ranage$





3.2.4 Bit-Plane Slicing

Pixel values are represented by unsigned integers which are composed of bits. For example, a grayscale image is represented by 8-bit unsigned integer. Every bit in this representation of pixel intensity contains some information. Instead of slicing on intensity levels, we can slice the pixels on their bit levels. We can consider 8-bit grayscale image as a plane that consists of bits. For example, lowest plane contains 0th bit of each pixel intensity, next higher plane contains 1st bit and the topmost plane contains 8th bit as shown in figure:



we can extract the 8th bit of all pixels and form an image similarly we can extract all 7th bits and form an image.

An image formed using nth bit from each pixel is called the nth bit plane. In case of images with 8-bit intensity values, there will be 8-bit planes.

Example: Bit-plane calculation

Nth bit plane is extracted by taking logical and with nth bit of each pixel. Let if binary value is 1101 1110 and we want to get 3rd bit plane, we will compute as follows:

$$\begin{array}{rl} 1101\ 1110 & \\ 0000\ 0100 & \& \\ \hline 0000\ 0100 & \end{array}$$

Let there is an image consisting of 4 pixels and their values are as under

Decimal Intensity values	**50**	**101**	**222**	**255**
Binary values	0011 0010	0110 0101	1101 1110	1111 1111
1st Bit plane	0000 0000	0000 0001	0000 0000	0000 0001
2nd Bit plane	0000 0010	0000 0000	0000 0010	0000 0010
…	…	…	…	…
8th bit plane	0000 0000	0000 0000	1000 0000	1000 0000

We can see that most significant bit contains highest value of information and least significant bit contains least information in the image.

.

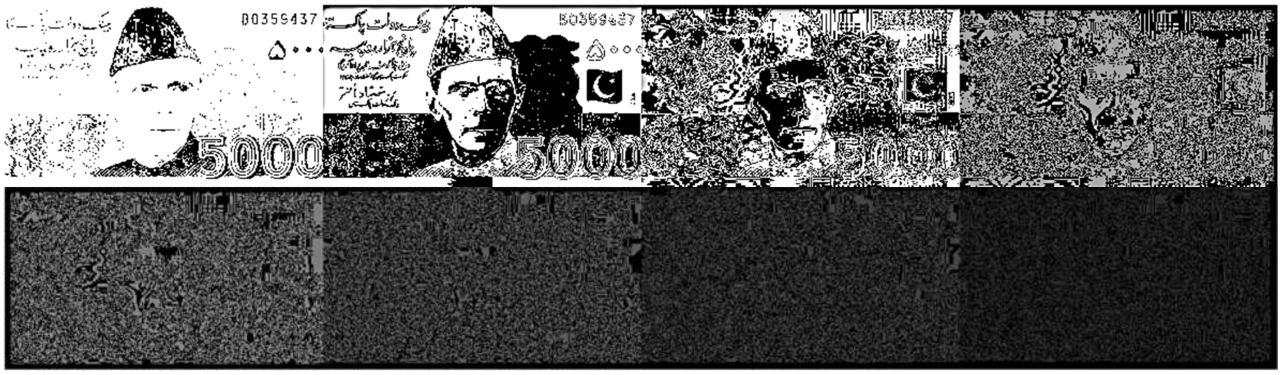


Figure ***Error! Use the Home tab to apply 0 to the text that you want to appear here.****-2: Bit-planes*

Reference Programs
***Program P3-10**: Single level image thresholding*
***Program P3-11**: Contrast stretching*
***Program P3-12**: Multi-level thresholding*
***Program P3-13**: Bit-plane slicing*

3.3 Histogram Processing

A histogram is a graphical representation that shows intensity levels and number of pixels that lie on each intensity level. Histogram can be defined in different ways. Image histogram shows the intensity distribution in an image. Let for a given image

- There are L levels ranging from $0 \rightarrow L-1$
- There are M rows and N columns and total number of pixels in image are MN
- There are n_k pixels for k intensity levels then
- Histogram bins: There will be $L-1$ histogram bins $n_0, n_1, n_2, \ldots., n_{L-1}$

A histogram made with such data is called the $un - normalized$ histogram. Here is program to calculate histogram of an input image.

API	Description
x.astype()	This function converts the data type of x as specified in its parameters
plt.plot(arr1,arr2)	This method plots the values in plane. $arr1$ refers to x-axis values and $arr2$ refers to y-axis values.
plt.vlines(x, ymin, ymax)	This method draws vertical line in plot on x location of x-axis to y location on y-axis. Height of line is determined by ymin and ymax values.

And here is the program that uses these APIs.

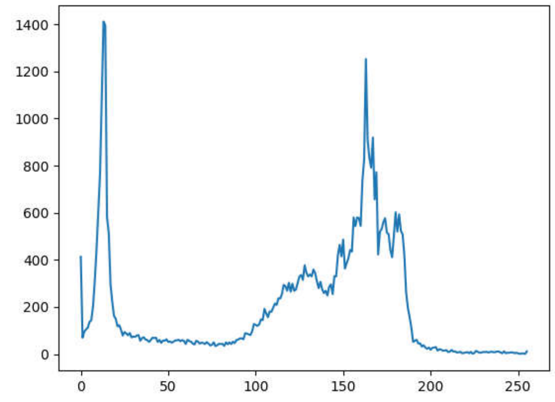
P3-2: Image Histogram – ImageHistogram.py

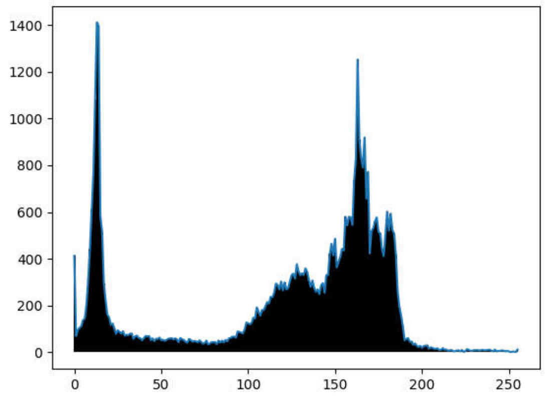
```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
img=cv.imread("cameraman.jpg")
img1=img[:,:,0]
bins=np.zeros(256)
for i in range(256):
      bins[i]=np.sum(img1==i)
print(bins.astype(np.uint8))
levels=range(256)
plt.plot(levels,bins)
for i in range(256):

```

```
14          plt.vlines(x=i, ymin=0, ymax=bins[i])
    plt.show()
```

Output





Program Description

- ***Line*6**: We create an array of 256 elements populated with zeros.
- ***Line*7**: for loop that iterates for 256 times
- ***Line*8**: We find the pixels that have intensity value equal to ***i*** and sum them up to find total number of pixels containing same value and store this count in ith bin.
- ***Line*9**: convert the bins datatype to unsigned integer and print its values
- ***Line*10**: create an array containing elements from 0-255
- ***Line*11**: plot levels and bins, if we do not create the vertical lines, output plot generated is shown in left figure.
- ***Line*13**: We create 256 vertical lines to show the bins
- ***Line*14**: plt.show() command is required to display the generated graph

In program P3-2, we have created histogram using python however OpenCV also provides API for creating image histogram.

API	Description
cv2.calcHist()	**cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]]).** This method takes 5 arguments. 1. **images** : it is the source image of type uint8 or float32. it should be given in square brackets, ie, "[img]". 2. **channels :** it is also given in square brackets. It the index of channel for which we calculate histogram. For example, if input is grayscale image, its value is [0]. For color image, you can pass [0],[1] or [2] to calculate histogram of blue, green or red channel respectively. 3. **mask :** mask image. To find histogram of full image, it is given as "None". But if you want to find histogram of particular region of image, you have to create a mask image for that and give it as mask. (I will show an example later.) 4. **histSize** : this represents our BIN count. Need to be given in square brackets. For full scale, we pass [256]. 5. **ranges** : this is our RANGE. Normally, it is [0,256].
plt.xlim()	This matplotlib function defines the range of values to be used on x-axis against histogram values

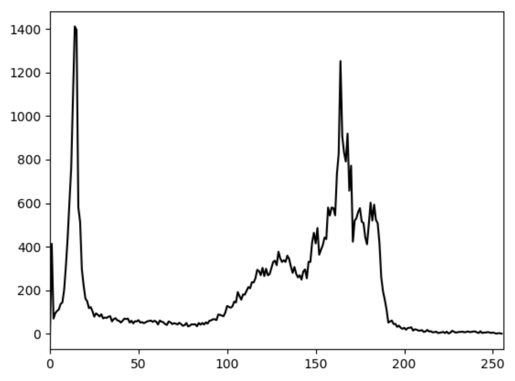
1

2 And here is the program that shows the use of this API

P3-3: Image Histogram – ImageHistogram.py

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
img=cv.imread("cameraman.jpg")
img1=img[:,:,0]
hist = cv.calcHist(images=[img1],
channels=[0],mask=None,histSize=[256],ranges=[0,255])
plt.plot(hist, color="black")
plt.xlim([0, 256])
plt.show()
```

Output



Program Description

- ***Line*6, 7**: use of OpenCV histogram calculation method.
- ***Line*9**: Specifying the range of values on x-axis in plot diagram

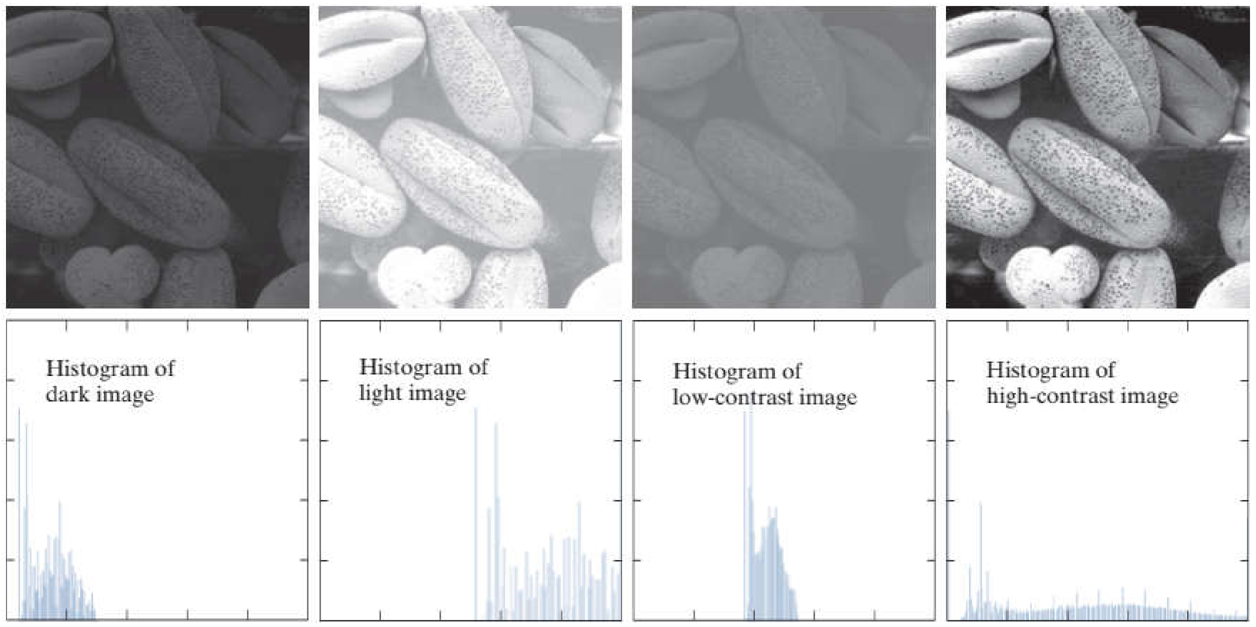
A histogram can be normalized in order to keep its values in the range of 0-1.

$p_r(k) = \frac{n_k}{MN}$ **(3 − 4)** and $\sum_k p_r(k) = 1$ **(3 − 5)**

You can see that this normalization process generates probability of each intensity level. Histogram shape is very much related to the appearance of image:

- *Left shifted*: if a histogram is more on left and less on right side, such an image will be darker
- *Right shifted*: if a histogram is right shifted, most of the pixels will have brighter values and image will be more bright
- *Center located*: if intensity distribution is more located in center and less on both sides the image will be of low contrast
- *Equally distributed*: if histogram is equally distributed, image will have high contrast and better appearance.

Following figures shows these points visually:



It shows that we should expand the histogram of an image over all available range of intensities in order to get a high-quality image.

<Write a program to shift image histogram to different locations and show the shifted image>

An image histogram could be processed as a whole or part. First two methods, we discuss here are global histogram processing methods whereas third is local histogram processing.

Histogram equalization (Contrast Stretching)

As discussed in previous section, there may be the possibility that when an image is viewed as histogram, it is squeezed to some part of complete histogram range. The process of histogram equalization spreads the squeezed histogram over all available range.

Let r be intensity of input image pixel where $r = 0,1,2,3, \ldots, L-1$. Intensity transformation function is given by

$$s = T(r) \text{ where } 0 \le r \le L-1 \qquad \mathbf{(3-6)}$$

Before explaining the histogram equalization, two concepts are required to understand.

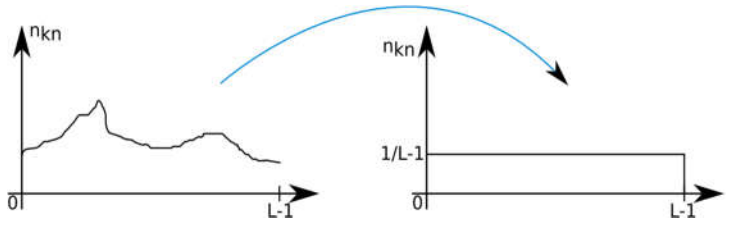
- $Probability\ density\ function\ (pdf)$: It is a function that tells the density (number of pixels) of particular intensity level i.e. probability.

$$p_k = \frac{n_k}{MN} = \frac{number\ of\ pixels\ with\ intensity\ k}{total\ number\ of\ pixels} \qquad \mathbf{(3-7)}$$

- $Cumulative\ distribution\ function\ (cdf)$: it is the function that tells total density till level k.

$$cdf = \sum_{i=0}^{k} p_i = \sum_{i=0}^{k} \frac{n_k}{MN} = \frac{number\ of\ pixels\ with\ intensity\ values\ from\ 0-k}{total\ number\ of\ pixels} \qquad \mathbf{(3-8)}$$

Idea behind histogram equalization is very simple. CDF of a continuous function is always one. We convert non-uniform cdf into uniform cdf as shown below:



Let for output image, if its CDF=1, it must cover all intensity levels i.e. $L-1$. This can be written mathematically as:

$$s_{L-1} = (L-1)\sum_{0}^{L-1} p_i \qquad \mathbf{(3-9)}$$

(The last level in output image must map to CDF of complete image)

Similarly, the first level of output image must map to first value of CDF

$$s_0 = (L-1)\sum_{0}^{0} p_i \qquad \mathbf{(3-10)}$$

And the output should be uniformly distributed over all range of available intensity levels so we can use same formula for all intensity levels.

$$s_k = (L-1)\sum_{0}^{k} p_i \qquad \mathbf{(3-11)}$$

This is intuitive explanation of histogram equivalized distribution an Interested reader may consult the append-A for more mathematical explanation of this process. <Write Append-A>

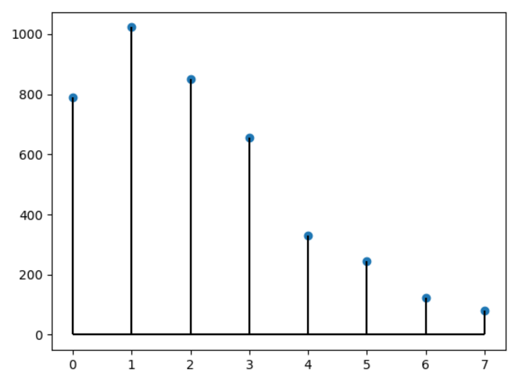
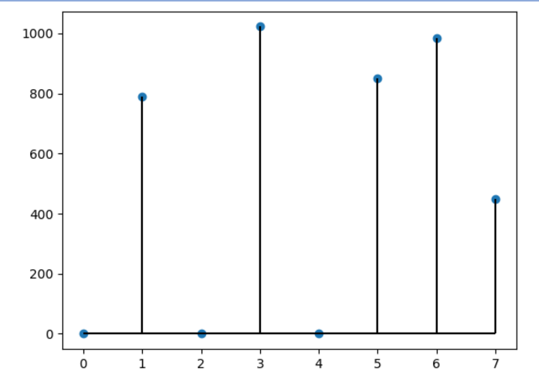
Histogram Equalization Example:

Let we have an image with following features:

- Intensity depth: $3-bit$
- No. of intensity levels: $8\ (0-7)$
- Image spatial resolution: $64x64 \rightarrow\ 4096$
- Histogram equalization formula: $Equation\ \mathbf{(3-11)}$

Intensity level r_k	Intensity density n_k	Probability density function $p_r(r_k) = \frac{n_k}{MN}$	$s_k = T(r_k)$ = $(L-1)\sum_0^k p_i$	$Round(s_k)$	n_{s_k}
$r_0 = 0$	790	$790/4096 \rightarrow 0.19$	$7*(0.19) = 1.33$	1	790
$r_1 = 1$	1023	0.25	$7*(0.19+0.25) = 3.08$	3	1023
$r_2 = 2$	850	0.21	**4.55**	**5**	**850**
$r_3 = 3$	656	0.16	**5.67**	**6**	**656+329 =985**
$r_4 = 4$	329	0.08	**6.23**	**6**	
$r_5 = 5$	245	0.06	**6.65**	**7**	**245+122 +81 = 448**
$r_6 = 6$	122	0.03	**6.86**	**7**	
$r_7 = 7$	81	0.02	**7.00**	**7**	

Now we create histograms of original and equalized images follows:

Before Equalization	After Equalization
	

<Write a program to perform the histogram equalization using python only>

P3-4: Image Histogram – ImageHistogram.py

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
levels=np.array([0,1,2,3,4,5,6,7])
input_image=np.array([790,1023,850,656,329,245,122,81])
```

```
6  output_img=np.zeros(len(input_image))
7  pdfs=input_image/4096
8  cdfs=np.zeros([len(pdfs)])
9  for i in range(len(pdfs)):
10     for j in range(0,i+1):
11         cdfs[i]+=pdfs[j]
12 cdfs=7*cdfs
13 cdfs=np.round(cdfs)
14 for i in range(len(input_image)):
15      output_img[i]+=np.sum(input_image[cdfs==i])
16 np.set_printoptions(precision=2)
17 print("Input::",input_image)
18 print("pdfs ::",pdfs)
19 print("cdfs ::",cdfs)
20 print("output::",output_img)
```

Output ::

```
Input:: [ 790 1023  850  656  329  245  122   81]
pdfs :: [0.19 0.25 0.21 0.16 0.08 0.06 0.03 0.02]
cdfs :: [1. 3. 5. 6. 6. 7. 7. 7.]
output:: [   0.  790.    0. 1023.    0.  850.  985.  448.]
```

Program Description

- ***Line*6, 7**: use of OpenCV histogram calculation method.
- ***Line*9**: Specifying the range of values on x-axis in plot diagram

OpenCV also provides API to perform histogram equalization.

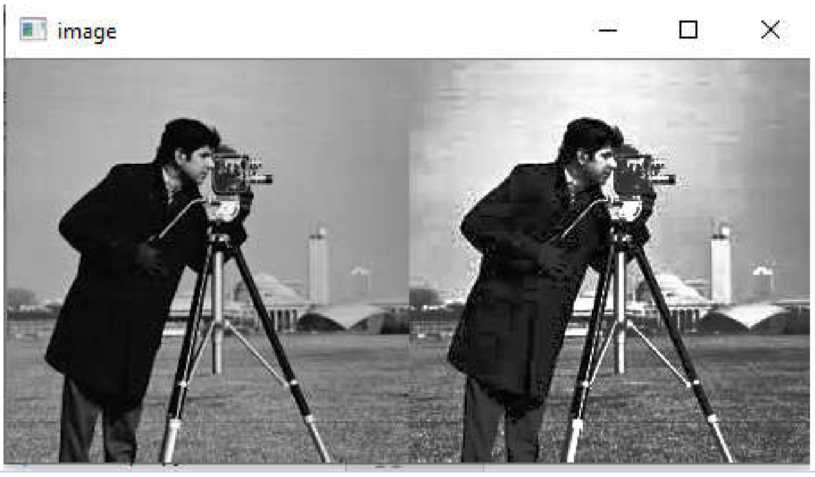
API	Description
cv2.EqualizeHist()	cv2.EqualizeHist(source) This method takes 1 argument. 1. **source**: it is the source single channel image of type uint8. It returns histogram equalized image
np.hstack()	**numpy.hstack(tup)** Stack arrays in sequence horizontally (column wise). This method takes 5 arguments. 1. **tup** : it is sequence of ndarrays. The arrays must have the same shape along all but the second axis, except 1-D arrays which can be any length. **Returns**: The array formed by stacking the given arrays.
cv.destroyAllWindows()	The function `destroyAllWindows` destroys all of the opened HighGUI windows.

And here is the implementation of this API

P3-5: Histogram Equalization – HistogramEqualization.py

```
1  import numpy as np
2  import cv2 as cv
3  import matplotlib.pyplot as plt
4  img=cv.imread("cameraman.jpg",0)
5  equ = cv.equalizeHist(img)
6  res = np.hstack((img, equ))
7  cv.imshow("image", res)
8  cv.waitKey(0)
9  cv.destroyAllWindows()
```

Output ::



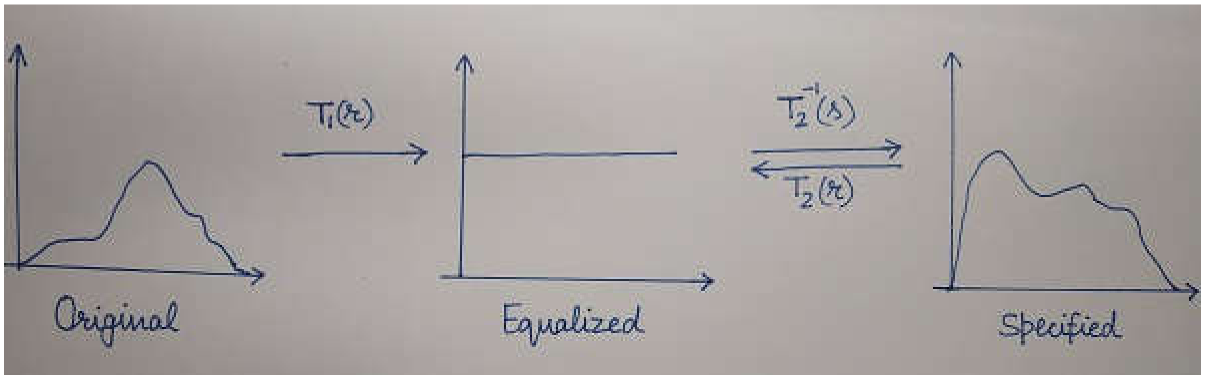
Program Description

- ***Line4***: reading an image as single channel image
- ***Line5***: Equalizing the histogram of image passed as parameter
- ***Line6***: It stackes the two images horizontally
- ***Line9***: destroying all image windows

<write a program in python to concatenate two images horizontally and vertically and display the results>

Histogram Matching (Specification)

There are the situations where histogram equalization is not the requirement however the need to transform the histogram of input image according to histogram of a reference image. This processing of matching of two histograms is called the histogram specification.



Let s and r are the output and input intensities respectively then transformation is written as:

$$s_k = T(r_k) = (L-1)\sum_{i=0}^{k} p_i \qquad \mathbf{(3-12)}$$

As before, we suppose that $p_r(r)$ is pdf of input image that we want to match with $p_z(z)$, the reference pdf. Then we can write

$$G(z) = s = T(r) \qquad \mathbf{(3-13)} \quad \text{and}$$

$$z = G^{-1}(s) = G^{-1}(T(r)) \qquad \mathbf{(3-14)}$$

We can write these equations more specifically as:

$$G(z_q) = (L-1)\sum_{i=0}^{q} p_z(z_i) \qquad \mathbf{(3-15)}$$

For a value of q that

$$G(z_q) = s_k \qquad \mathbf{(3-16)} \quad \text{and}$$

$$z_q = G^{-1}(s_k) \qquad \mathbf{(3-17)}$$

Considering these two relationships, we can design the histogram matching algorithm as follows:

1. Compute the histogram $p_r(r)$ of the input image
2. Equalize this histogram using equation $s_k = T(r_k) = (L-1)\sum_{i=0}^{k} p_r\,(r_i)$ and round the output to integer values
3. Compute the values of $G(z_q)$ by using the relationship $G(z_q) = (L-1)\sum_{i=0}^{q} p_z\,(z_i)$ where $p_z(z_i)$ is the values of specified histogram. Round the value of G.
4. For every value of s_k, use the stored values of G from step 3 such that $G(z_q)$ is closest to s_k. When more than one value of z_q gives the same match, choose the smallest value by convention. Form the output image by mapping every equalized pixel s_k to corresponding z_q using step 4
5. Compute the output image pdf

The procedure is explained below with the help of example.

<u>Example</u>

Let we have an image with following features:

- Intensity depth: $3-bit$
- No. of intensity levels: $8\ (0-7)$
- Image spatial resolution: $64x64 \rightarrow\ 4096$

Following tables shows the details of input and reference histograms

1	2	3	4	5	6	7	8	9
$\boldsymbol{r_k}$	n_k	$\boldsymbol{p_r k = \frac{n_k}{MN}}$	$s_k = T(r_k)$	$Round(\boldsymbol{s_k})$	$\boldsymbol{n_{s_k}}$	Specified $pdf = p_z(k)$	$\boldsymbol{G(z_q)}$ =$\boldsymbol{(L-1)\sum_0^k p_i}$	$\boldsymbol{G(z_q)}$
$\boldsymbol{r_0 = 0}$	790	**0.19**	1.33	**1**	790	0.00	0.00	0
$\boldsymbol{r_1 = 1}$	1023	**0.25**	3.08	**3**	1023	0.00	0.00	0
$\boldsymbol{r_2 = 2}$	850	**0.21**	**4.55**	**5**	**850**	**0.00**	**0.00**	**0**
$\boldsymbol{r_3 = 3}$	656	**0.16**	**5.67**	**6**	**985**	**0.15**	**7*(0.15)**	**1**

							=1.05	
$r_4 = 4$	329	0.08	6.23	6		0.20	2.45	2
$r_5 = 5$	245	0.06	6.65	7	448	0.30	4.55	5
$r_6 = 6$	122	0.03	6.86	7		0.20	5.95	6
$r_7 = 7$	81	0.02	7.00	7		0.15	7.00	7

1. Compute the pdf $p_r(r)$ for input image which is given in column 3 in above table
2. Equalize the histogram and find values that are given in column 5 in above table.
3. Compute the function $G(z_q)$ by considering the specified histogram as shown in column 8 and its rounded values are given in column 9
4. For every value of s_k, use the stored values of G from step 3 such that $G(z_q)$ is closest to s_k as shown in column 17
5. Output image pdf is shown in column 18.

10	11	12	13	14	15	16	17	18
r_k	n_k	s_k	n_{s_k}	Z_q	Specified pdf	$G(z_q)$	$n_z k$	$p_z(z_i)$
$r_0 = 0$	790	1	790	$Z_0 = 0$	0.00	0	0	0/4096=0
$r_1 = 1$	1024	3	1024	$Z_1 = 1$	0.00	0	0	0
$r_2 = 2$	850	5	850	$Z_2 = 2$	0.00	0	0	0
$r_3 = 3$	656	6	985	$Z_3 = 3$	0.15	1	790	0.19
$r_4 = 4$	329	6		$Z_4 = 4$	0.20	2	1024	0.25
$r_5 = 5$	245	7	448	$Z_5 = 5$	0.30	5	850	0.21
$r_6 = 6$	122	7		$Z_6 = 6$	0.20	6	985	0.24
$r_7 = 7$	81	7		$Z_7 = 7$	0.15	7	448	0.11

Following table lists new APIs that we are going to use in next program:

API	Description
x.astype()	`x.astype(new type)` This is numpy library method that converts data type of object “x”. We provide new data type as parameter. For example, in our program we provide “np.uint8”, 8-bit unsigned integer
x.__contains__():	x.__contains__(value): This method takes one value as parameters and return true or false depending upon “x” contains that value or not.

And here is program

P3-5: Histogram Matching – HistogramMatching.py

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

r_k=np.array([0,1,2,3,4,5,6,7])
n_k=np.array([790,1023,850,656,329,245,122,81])
pr_k=n_k/4096
s_k=np.zeros(len(n_k))          # for s_k=T(r_k)
```

```python
n_s_k=np.zeros(len(n_k))           # pixel mapping n_k --> n_s_k
def histEqualization():
    s_k=np.zeros([len(n_k)])
    for i in range(len(n_k)):
        for j in range(0,i+1):
            s_k[i]+=pr_k[j]
    s_k=7*s_k
    s_k=np.round(s_k)
    for i in range(len(n_k)):
         n_s_k[i]+=np.sum(n_k[s_k==i])
    return n_s_k.astype(np.uint8),s_k.astype(np.uint8)

n_s_k,s_k=histEqualization()
pz_k=np.array([0.00,0.00,0.00,0.15,0.20,0.30,0.20,0.15])
Gz_q=np.zeros([len(pz_k)])
for i in range(len(pz_k)):
    for j in range(0,i+1):
        Gz_q[i]+=pz_k[j]
Gz_q=7*Gz_q
Gz_q=np.round(Gz_q).astype(np.uint8)
nz_k=np.zeros(len(n_k))
s_k_missing=[]
Gz_k_missing=[]
for i in range(len(n_k)):
     ind_gz=Gz_q[i]
     ind_sk=s_k[i]
     if s_k.__contains__(ind_gz):
         val_nk=n_k[s_k==ind_gz]
         nz_k[i]=sum(val_nk)
     if not(s_k.__contains__(ind_gz)) and not(Gz_k_missing.__contains__(ind_gz)):
          Gz_k_missing.append(ind_gz)
     if not(Gz_q.__contains__(ind_sk))and not(s_k_missing.__contains__(ind_sk)):
         s_k.__contains__(ind_gz): (ind_sk)

s_k_missing=np.asarray(s_k_missing).astype(np.uint8)
for i in range(len(s_k_missing)):
    forward=0
    backward=0
    for j in range(np.round(len(Gz_q)/2).astype(np.uint8)):
        forward=s_k_missing[i]+j
        backward=s_k_missing[i]-j
        if(Gz_q[forward]+j==s_k_missing[i]):
            nz_k[forward]+=n_k[np.where(s_k==s_k_missing)]
            break
        elif (Gz_q[backward]-j==s_k_missing[i]):
            nz_k[backward]+=n_k[np.where(s_k==s_k_missing)]
            break
pz_z=nz_k/4096
np.set_printoptions(precision=2)
print("r_k::",r_k)
print("n_k::",n_k)
print("n_k::",pr_k)
print("s_k::",s_k)
print("n_s_k::",n_s_k)
print("pz_k::",pz_k)
print("Gz_q::",Gz_q)
print("nz_k::",nz_k)
print("pz_z::",pz_z)
```

Output ::

```
r_k:: [0 1 2 3 4 5 6 7]
```

```
n_k:: [ 790 1023  850  656  329  245  122   81]
n_k:: [0.19 0.25 0.21 0.16 0.08 0.06 0.03 0.02]
s_k:: [1 3 5 6 6 7 7 7]
n_s_k:: [  0  22   0 255   0  82 217 192]
pz_k:: [0.   0.   0.   0.15 0.2  0.3  0.2  0.15]
Gz_q:: [0 0 0 1 2 5 6 7]
nz_k:: [   0.    0.    0.  790. 1023.  850.  985.  448.]
pz_z:: [0.   0.   0.   0.19 0.25 0.21 0.24 0.11]
```

Program Description

We leave the explain of this program as user is expected to understand it himself

<change above program to descriptive names instead of mathematical symbols like "n_r" could be replaced with "levels", "n_k→pixel_count"…>

<write a program to match the histogram based on images>

Local Histogram Processing (Local Histogram Equalization-LHE)

The methods we have studied till now are based on whole image histogram processing. In other words

$$\bigvee_{j=0}^{MN} s = T(r) = (L-1)\sum_{i}^{k} p_r(k) \qquad \mathbf{(3-18)}$$

Sometimes, it is required to process parts of image in order to extract the hidden information.

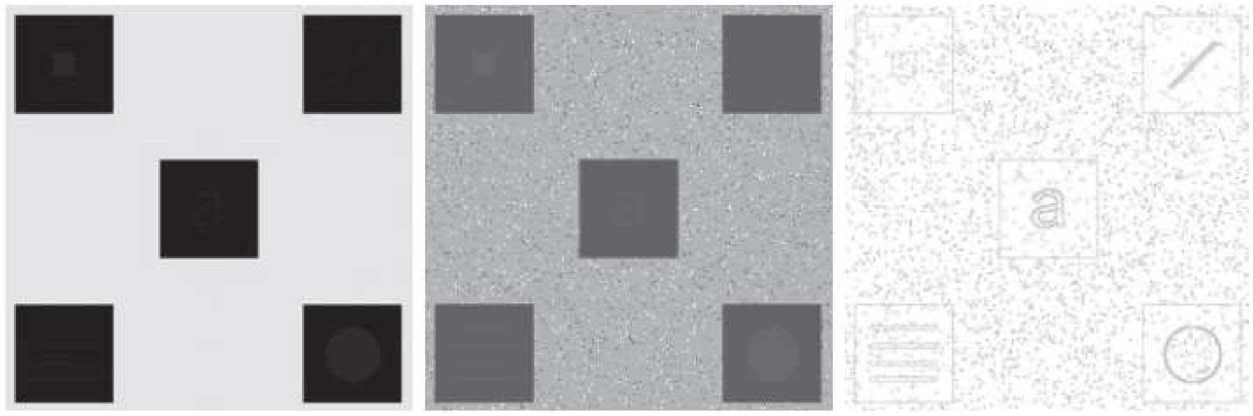


Figure Error! Use the Home tab to apply 0 to the text that you want to appear here.**-3: A) Original Image, B) Global Histogram Equalization C) Local Histogram Processing**

The local histogram processing is done by defining a window size and moving it pixel to pixel in horizontal and vertical directions. At each location, the histogram of the points in the window is computed, and either a *histogram equalization* or *histogram specification* transformation function is obtained. This function is used to map the intensity of the pixel centered in the neighborhood. The center of window is then moved one pixel to right and the transformation is computed again. When the column-wise moving is complete then window center is placed again on next row and first column.

This process is also known as adaptive histogram processing (AHE). It differs from ordinary histogram equalization in the respect that the adaptive method computes several histograms, each corresponding to a distinct region (window) of the image, and uses them to redistribute the lightness values of the image. It is therefore suitable for improving the local contrast and enhancing the definitions of edges in each region of an image.

<local histogram processing example code
https://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_local_equalize.html>

However, AHE has a tendency to overamplify noise in relatively homogeneous regions of an image. A variant of adaptive histogram equalization called contrast limited adaptive histogram equalization (CLAHE) prevents this by limiting the amplification.

In this, image is divided into small blocks called "tiles" (tileSize is 8x8 by default in OpenCV). Then each of these blocks are histogram equalized as usual. So, in a small area, histogram would confine to a small region (unless there is noise). If noise is there, it will be amplified. To avoid this, contrast limiting is applied. *If any histogram bin is above the specified contrast limit (by default 40 in OpenCV), those pixels are clipped and distributed uniformly to other bins before applying histogram equalization.*

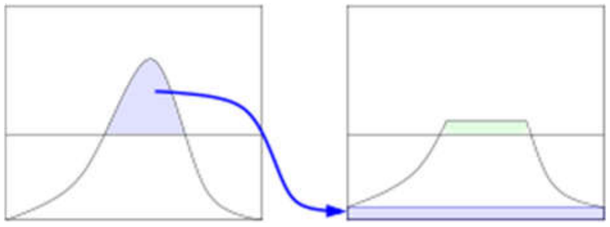


Figure Error! Use the Home tab to apply 0 to the text that you want to appear here.**-4: Contrast bin clipping**

After equalization, to remove artifacts in tile borders, bilinear interpolation is applied. Below code snippet shows how to apply CLAHE in OpenCV:

API	Description
cv.createCLAHE()	`cv.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))` This method gets two parameters: • `clipLimit`: Threshold for contrast limiting • `tileGridSize:` Size of grid for histogram equalization. Input image will be divided into equally sized rectangular tiles. tileGridSize defines the number of tiles in row and column.
clahe.apply()	**clahe.apply(img):** This method gets one parameter, the image, Equalizes the histogram of a grayscale image using Contrast Limited Adaptive Histogram Equalization.

And here is the program

P3-5: Histogram Matching – HistogramMatching.py

```
import cv2 as cv
img=cv.imread("cameraman.jpg",0)
clahe = import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np

img=cv.imread("cameraman.jpg",0)
clahe = cv.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
```

```
13  cl1 = clahe.apply(img)
14  cv.imwrite('clahe_2.jpg',cl1)
15  cv.imshow("cameraman",img)
16  cv.imshow("",cl1)
17  cv.waitKey(0)
18
19
```

Output ::

Program Description

Line 13: It computes the CLAHE on image
Line 14: It applies the computed CLAHE on image

Histogram statistics and image enhancement

Some statistics associated with an image are listed below:

Statistic	Formula
Mean	$m = \sum_{0}^{L-1} r_i p(r_i) \qquad (\mathbf{3}-19)$
First moment	$\mu_1 = \sum_{0}^{L-1} (r_i - m) p(r_i) \qquad (\mathbf{3}-20)$
Second Moment/ Variance $\boldsymbol{\sigma^2}$	$\mu_2 = \sum_{0}^{L-1} (r_i - m)^2 p(r_i) \qquad (\mathbf{3}-21)$
Nth moment	$\mu_n = \sum_{0}^{L-1} (r_i - m)^n p(r_i) \qquad (\mathbf{3}-22)$

Mean and variance of the image are the values that can be used for contrast enhancement. These two quantities can be used either at global or local level. When used on global level, m and $\sigma^2$are calculated for entire image and in case of local processing, these quantities are calculated for part of image. We can describe the m and σ^2 for local image areas as:

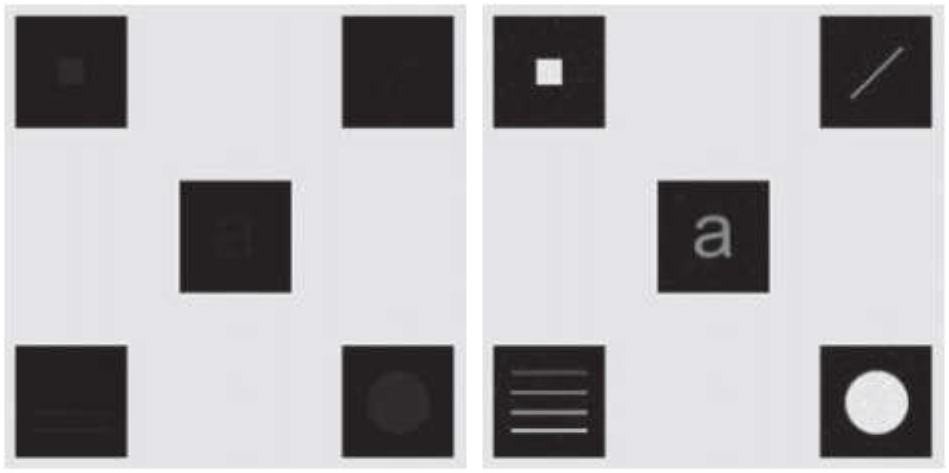
$$m_{s_{xy}} = \sum_{0}^{L-1} r_i p_{s_{xy}}(r_i) \qquad (\mathbf{3}-23)$$

Here S_{xy} defines the neighborhood of window size of image part. And variance is given as:

$$\mu_{s_{xy}} = \sum_{0}^{L-1} \left(r_i - m_{s_{xy}}\right)^2 p_{s_{xy}}(r_i) \quad (\mathbf{3}-24)$$

<u>**Example**</u>

The problem at hand is to enhance the low contrast detail in the dark areas of the image, while leaving the light background unchanged. A method used to determine whether an area is relatively light or dark at a point (x, y) is to compare the average local intensity. Firstly, find all four statistics $m, m_{s_{xy}}, \sigma^2, \sigma^2_{s_{xy}}$



- A pixel $p(x,y)$ is candidate for processing if $k_0 m \le m_{s_{xy}} \le k_1 m$ for some non-negative value of k where $k_0 \le k$_1. We choose k arbitrary. For example, if our focus is on areas that are darker than one-quarter of the mean intensity we would choose k0 = 0 and k1 = 0.25.
- A pixel as candidate for processing if $k_2\sigma \le \sigma_{s_{xy}} \le k_3\sigma$. For example, to enhance a dark area of low contrast, we might choose k2 = 0 and k3 = 0.1.

A pixel that meets all the preceding conditions for local enhancement is processed by multiplying it by a specified constant, C, to increase (or decrease) the value of its intensity level relative to the rest of the image. Pixels that do not meet the enhancement conditions are not changed. Mathematically this procedure can be written as following formula

$$g(x,y) = f(x) = \begin{cases} cf(x,y), & if \quad k_0 m \le m_{s_{xy}} \le k_1 m \ \textbf{\textit{and}} \ \ k_2\sigma \le \sigma_{s_{xy}} \le k_3\sigma \\ f(x,y), & otherwise \qquad (\mathbf{3}-25) \end{cases}$$

The Spatial Filtering

An image is a matrix of values. The operations that are performed on images area also matrix operations. However new terms and names may be given to operations and operator matrices.

- *Filter*: A special type of matrix that is operated with image. A filter is also known as *kernel*, *mask*, *template and window*.
- *Linear Filter*: If the operation performed on image pixels is linear then operator matrix is called linear filter.
- *Non − Linear Filter*: If the operator filter (matrix) performs the operation on image pixels that is non-linear then the filter is called non-linear filter.

A linear filter performs the *sum of products* operation and a filter kernel. Usually a kernel defines the neighborhood of reference pixel. Let we consider an image f of size 5x5 and kernel w of size 3x3 then general sum of products operation can be defined as:

Table 1: A) Filter Kernel w B) Image F

W(0,0)	W(0,1)	W(0,2)
W(1,0)	W(1,1)	W(1,2)
W(2,0)	W(2,1)	W(2,2)

F(0,0)	F(0,1)	F(0,2)	F(0,3)	F(0,4)
F(1,0)	F(1,1)	F(1,2)	F(1,3)	F(1,4)
F(2,0)	F(2,1)	F(2,2)	F(2,3)	F(2,4)
F(3,0)	F(3,1)	F(3,2)	F(3,3)	F(3,4)
F(4,0)	F(4,1)	F(4,2)	F(4,3)	F(4,4)

The filter maps over specific position on image which is shown by dark solid line in image. The operation is defined with reference to window center pixel, it is denoted as:

$$g(x,y) = f(0,0)*w(0,0) + f(0,1)*w(0,1) + f(0,2)*w(0,2) +$$
$$f(1,0)*w(1,0) + f(1,1)*w(1,1) + f(1,2)*w(1,2) +$$
$$f(2,0)*w(2,0) + f(2,1)*w(2,1) + f(2,2)*w(2,2)$$
$$= \sum_{x=0}^{2}\sum_{y=0}^{2} f(x,y)*w(x,y) \qquad (3-26)$$

If we consider the center pixel as reference pixel then it coordinates will be (0,0). Then our matrices could be as follows:

Table 2: A) Filter Kernel w B) Image F

W(-1,-1)	W(0,-1)	W(1,-1)
W(-1,0)	W(0,0)	W(1,0)
W(-1,1)	W(0,1)	W(1,1)

F(-1,-1)	F(0,-1)	F(1,-1)	F(0,3)	F(0,4)
F(-1,0)	F(0,0)	F(1,0)	F(1,3)	F(1,4)
F(-1,1)	F(0,1)	F(1,1)	F(2,3)	F(2,4)
F(3,0)	F(3,1)	F(3,2)	F(3,3)	F(3,4)
F(4,0)	F(4,1)	F(4,2)	F(4,3)	F(4,4)

And resulting equation will be:

$$g(x,y) = f(-1,-1)*w(-1,-1) + f(-1,0)*w(-1,0) + f(-1,1)*w(-1,1) +$$
$$f(-1,0)*w(-1,0) + f(0,0)*w(0,0) + f(1,0)*w(1,0) +$$
$$f(-1,1)*w(-1,1) + f(0,1)*w(0,1) + f(1,1)*w(1,1)$$
$$= \sum_{x=-1}^{1}\sum_{y=-1}^{1} f(x,y)*w(x,y) \qquad (3-27)$$

As coordinates x and y are varied, the center of the kernel moves from pixel to pixel, generating the filtered image, g, in the process. As an example, for next processing the matrices will be like this:

W(-1,-1)	W(0,-1)	W(1,-1)
W(-1,0)	W(0,0)	W(1,0)
W(-1,1)	W(0,1)	W(1,1)

F(0,0)	F(-1,-1)	F(0,-1)	F(1,-1)	F(0,4)
F(1,0)	F(-1,0)	F(0,0)	F(1,0)	F(1,4)
F(2,0)	F(-1,1)	F(0,1)	F(1,1)	F(2,4)
F(3,0)	F(3,1)	F(3,2)	F(3,3)	F(3,4)
F(4,0)	F(4,1)	F(4,2)	F(4,3)	F(4,4)

Spatial Correlation and Convolution

Correlation and convolution are similar procedure except that in convolution, the filter is rotated by 180^0.

- $Correlation\ h \star f$: it is the measure of the similarity of two signals

$$g(x,y) = h \text{ ☆ } f = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) * f(x+s, y+t) \qquad (3-28)$$

- $Convolution$: it measures the effect of one signal on another signal

$$g(x,y) = h \star f = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) * f(x-s, y-t) \qquad (3-29)$$

We being by explaining 1-2 correlation. Let we have kernel of size 5 and signal (1D image) of size 8, the correlation and convolution are given as:

$$g(x) = w \text{ ☆} f = \sum_{-a}^{a} w(s) f(x+s) \qquad (3-30)$$

and

$$g(x) = w \star f = \sum_{-a}^{a} w(s) f(x-s) \qquad (3-31)$$

Example

Let we explain 1D correlation and convolution with the help of an example. Let $f = [2\ 1\ 3\ 1\ 0\ 1\ 2\ 1]$ and $w = [1\ 2\ 4\ 2\ 8]$

Correlation (w ☆ f)	**Convolution ($w \star f$)**
[2 1 3 1 0 1 2 1] [1 2 4 2 8] **We position the center of filter kernel at first element of image**	[2 1 3 1 0 1 2 1] [8 2 4 2 1]
[0 0 2 1 3 1 0 1 2 1 0 0] [1 2 4 2 8] **Pad zeros on both sides of signal to make operation valid. If we do not pad then there will be no value to which 1,2 will be multiplied. Padded values are shown as blue zeros.**	[0 0 2 1 3 1 0 1 2 1 0 0] [8 2 4 2 1]
[0 0 2 1 3 1 0 1 2 1 0 0] [1 2 4 2 8] --------------------------------- **0*1+0*2+2*4+1*2+3*8 = 0+0+8+2+24 = 34**	[0 0 2 1 3 1 0 1 2 1 0 0] [8 2 4 2 1] --------------------------------- **0*8+0*2+2*4+1*2+3*1 = 0+0+8+2+3 = 13**
[0 0 2 1 3 1 0 1 2 1 0 0] [1 2 4 2 8] --------------------------------- **0*1+2*2+1*4+3*2+1*8 = 0+4+4+6+8 = 22**	[0 0 2 1 3 1 0 1 2 1 0 0] [8 2 4 2 1] --------------------------------- **0*8+2*2+1*4+3*2+1*1 = 0+4+4+6+1 = 15**
[0 0 2 1 3 1 0 1 2 1 0 0] [1 2 4 2 8] --------------------------------- **2*1+1*2+3*4+1*2+0*8 = 2+2+12+2+0 = 18**	[0 0 2 1 3 1 0 1 2 1 0 0] [8 2 4 2 1] --------------------------------- **2*8+1*2+3*4+1*2+0*1 = 16+2+12+2+0 = 32**

[0 0 2 1 3 1 0 1 2 1 0 0] [1 2 4 2 8] --------------------------------- 1*1+3*2+1*4+0*2+1*8 = 1+6+4+0+8 = 19	[0 0 2 1 3 1 0 1 2 1 0 0] [8 2 4 2 1] --------------------------------- 1*8+3*2+1*4+0*2+1*1 = 8+6+4+0+1 = 19
[0 0 2 1 3 1 0 1 2 1 0 0] [1 2 4 2 8] --------------------------------- 3*1+1*2+0*4+1*2+2*8 = 3+2+0+2+16 = 23	[0 0 2 1 3 1 0 1 2 1 0 0] [8 2 4 2 1] --------------------------------- 3*8+1*2+0*4+1*2+2*1 = 24+2+0+2+2 = 30
[0 0 2 1 3 1 0 1 2 1 0 0] [1 2 4 2 8] --------------------------------- 1*1+0*2+1*4+2*2+1*8 = 1+0+4+4+8 = 17	[0 0 2 1 3 1 0 1 2 1 0 0] [8 2 4 2 1] --------------------------------- 1*8+0*2+1*4+2*2+1*1 = 8+0+4+4+1 = 17
[0 0 2 1 3 1 0 1 2 1 0 0] [1 2 4 2 8] --------------------------------- 0*1+1*2+2*4+1*2+0*8 = 0+2+8+2+0 = 14	[0 0 2 1 3 1 0 1 2 1 0 0] [8 2 4 2 1] --------------------------------- 0*8+1*2+2*4+1*2+0*1 = 0+2+8+2+0 = 12
[0 0 2 1 3 1 0 1 2 1 0 0] [1 2 4 2 8] --------------------------------- 1*1+2*2+1*4+0*2+0*8 = 1+4+4+0+0 = 9 We end the procedure when the center of filter kernel reaches the end point in image.	[0 0 2 1 3 1 0 1 2 1 0 0] [8 2 4 2 1] --------------------------------- 1*8+2*2+1*4+0*2+0*1 = 8+4+4+0+0 = 16
w☆***f***= [34 22 18 19 23 17 14 9]	***w*** ⋆ ***f***= [13 15 32 19 30 17 12 16]

Now we look at correlation and convolution in 2D. We write few steps of convolution only here, the reader can work out rest of the steps. Let our image f and filter w are:

$$W = \begin{bmatrix} 1 & 1 & 2 \\ 2 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \text{ and } f = \begin{bmatrix} 1 & 2 & 0 & 1 & 2 \\ 2 & 3 & 1 & 1 & 2 \\ 1 & 4 & 2 & 2 & 0 \\ 3 & 2 & 3 & 3 & 0 \\ 1 & 0 & 0 & 2 & 1 \end{bmatrix} \qquad \text{then}$$

$$\begin{bmatrix} 1 & 2 & 0 & 1 & 2 \\ 2 & 3 & 1 & 1 & 2 \\ 1 & 4 & 2 & 2 & 0 \\ 3 & 2 & 3 & 3 & 0 \\ 1 & 0 & 0 & 2 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 1 & 2 & 0 \\ 0 & 2 & 3 & 1 & 1 & 2 & 0 \\ 0 & 1 & 4 & 2 & 2 & 0 & 0 \\ 0 & 3 & 2 & 3 & 3 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 2 \\ 2 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \star \begin{bmatrix} \boxed{\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 2 & 3 \end{matrix}} & \begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 1 & 1 & 2 & 0 \end{matrix} \\ \begin{matrix} 0 & 1 & 4 \\ 0 & 3 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix} & \begin{matrix} 2 & 2 & 0 & 0 \\ 3 & 3 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{matrix} \end{bmatrix}$$

$$= (0*1+0*1+0*2)+(0*2+1*1+2*1)+(0*1+2*1+3*1) = (0+3+5) = 8$$

$$\begin{bmatrix}1&1&2\\2&1&1\\1&1&1\end{bmatrix}\star\begin{bmatrix}0&0&0&0&0&0&0\\0&1&2&0&1&2&0\\0&2&3&1&1&2&0\\0&1&4&2&2&0&0\\0&3&2&3&3&0&0\\0&1&0&0&2&1&0\\0&0&0&0&0&0&0\end{bmatrix}$$

$$= (0*1+0*1+0*2)+(1*2+2*1+0*1)+(2*1+3*1+1*1) = 0+4+6$$
$$= \mathbf{10}$$

$$\begin{bmatrix}1&1&2\\2&1&1\\1&1&1\end{bmatrix}\star\begin{bmatrix}0&0&0&0&0&0&0\\0&1&2&0&1&2&0\\0&2&3&1&1&2&0\\0&1&4&2&2&0&0\\0&3&2&3&3&0&0\\0&1&0&0&2&1&0\\0&0&0&0&0&0&0\end{bmatrix}$$

$$= (0*1+0*1+0*2)+(2*2+0*1+1*1)+(3*1+1*1+1*1) = 0+5+5$$
$$= \mathbf{10}$$

$$\begin{bmatrix}1&1&2\\2&1&1\\1&1&1\end{bmatrix}\star\begin{bmatrix}0&0&0&0&0&0&0\\0&1&2&0&1&2&0\\0&2&3&1&1&2&0\\0&1&4&2&2&0&0\\0&3&2&3&3&0&0\\0&1&0&0&2&1&0\\0&0&0&0&0&0&0\end{bmatrix}$$

$$= (0*1+0*1+0*2)+(0*2+1*1+2*1)+(1*1+1*1+2*1) = 0+5+4$$
$$= \mathbf{9}$$

$$\begin{bmatrix}1&1&2\\2&1&1\\1&1&1\end{bmatrix}\star\begin{bmatrix}0&0&0&0&0&0&0\\0&1&2&0&1&2&0\\0&2&3&1&1&2&0\\0&1&4&2&2&0&0\\0&3&2&3&3&0&0\\0&1&0&0&2&1&0\\0&0&0&0&0&0&0\end{bmatrix}$$

$$= (0*1+0*1+0*2)+(1*2+2*1+0*1)+(1*1+2*1+0*1) = 0+4+3$$
$$= \mathbf{7}$$

$$\begin{bmatrix}1&1&2\\2&1&1\\1&1&1\end{bmatrix}\star\begin{bmatrix}0&0&0&0&0&0&0\\0&1&2&0&1&2&0\\0&2&3&1&1&2&0\\0&1&4&2&2&0&0\\0&3&2&3&3&0&0\\0&1&0&0&2&1&0\\0&0&0&0&0&0&0\end{bmatrix}$$

$$= (0*1+1*1+2*2)+(0*2+2*1+3*1)+(0*1+1*1+4*1) = 5+5+5$$
$$= \mathbf{15}$$

The resulting matrix becomes like

$$\begin{bmatrix} 8 & 10 & 10 & 9 & 7 \\ 15 & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix}$$

<Do the rest of computations and fill the matrix>

<write a program in python to perform the correlation and convolution of an image and filter>

<you can get convolution idea from here

https://www.pyimagesearch.com/2016/07/25/convolutions-with-opencv-and-python/>

API	Description
cv.filter2D()	**`cv.filter2D(src, ddepth, kernel)`** This method convolves the kernel with input image. Its parameters are: • **`src`** – A Mat object representing the source (input image) for this operation. • **`ddepth`** – A variable of the type integer representing the depth of the output image • **`kernel`** – A Mat object representing the convolution kernel. **`Returns`**: • An image which is applied the filter

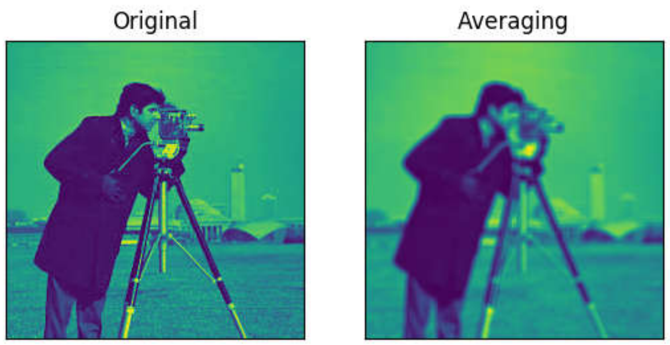
Here is an example program that uses this API.

P3-6: ConvolvingAnImage – HistogramMatching.py

```python
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

img=cv.imread("cameraman.jpg",0)
kernel = np.ones((5,5),np.float32)/25
dst = cv.filter2D(img,-1,kernel)
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(dst),plt.title('Averaging')
plt.xticks([]), plt.yticks([])
plt.show()
```

Output ::



Program Description

***Line* 13**: **`It computes the CLAHE on image`**
***Line* 14**: **`It applies the computed CLAHE on image`**

To find correlation, you can easily rotate a matrix by writing few lines of python code. Finding a convolution with rotated matrix is same as finding the correlation with non-rotated matrix.

P3-6: RotatingMatrix – HistogramMatching.py

```python
mat=np.random.randint(1,10,[5,5])
# reversing the matrix
rows,cols=mat.shape
rotated=np.zeros([5,5],np.uint8)
for i in range(rows):
    rotated[i,:]=mat[-(i+1),:]
for j in range(cols):
    rotated[:,j]=mat[:,-(j+1)]
print(mat)
print(rotated)
```

Output ::

```
Matrix:: [[7 4 2 9 3]        rotated:: [[3 9 2 4 7]
          [2 8 7 2 7]                   [7 2 7 8 2]
          [6 2 2 1 9]                   [9 1 2 2 6]
          [2 4 6 7 1]                   [1 7 6 4 2]
          [4 6 5 7 2]]                  [2 7 5 6 4]]
```

Correlation and convolution properties:

Property	Correlation	Convolution
Commutative	-	$f \star g = g \star f$
Associative	-	$f \star (g \star h) = f \star (g \star h)$
Distributive	$f☆(g+h) = (f☆g) + (f☆h)$	$f \star (g+h) = (f \star g) + (f \star h)$

<Write python program to verify these properties visually>

Convolution filters are also known as $convolution\ masks$ or $convolution\ kernel$.

Convolving with multiple kernels

Sometime there may be the condition where we need to convolve an image with kernel-1 and then its result with 2nd kernel and so on. Such a situation for n-kernels can be written as:

$$w_n \star \ldots w_3 \star w_2 \star (w_1 \star f) \quad (\mathbf{3}-31)$$

We can simplify the operation by some mathematical work on equation $(\mathbf{3}-31)$. We use the commutative property of convolution in this mathematical workaround.

$$\boldsymbol{w} = w_1 \star w_2 \star \ldots . \star w_n \qquad (\mathbf{3}-32).$$

and

$$convolution = \boldsymbol{w} \star f \qquad (\mathbf{3}-33).$$

Separable Kernel Filters

A 2D function is said to be separable if it can be written as product two 1D function. A kernel is also a function and if that can be written as product of two kernels, it is also called the separable kernel.

$$G(x,y) = G_1(x)G_2(y) \qquad (\mathbf{3}-34).$$

Example:

Let we have a kernel as follows:

$$w = G(x,y) = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \rightarrow G_1(x)G_2(y) \qquad (\mathbf{3}-35).$$

Separable kernels provide computational advantage in two ways:

- A product of two 1D kernels is equal to convolution of these vectors.
- Associative property of convolution. We can process the kernels and images in any order and by selecting suitable processing order, we can reduce the number of basic operations (multiplication and addition).

$$\boldsymbol{w} \star f = (w_1 \star w_2) \star f = w_1 \star (w_2 \star f) = (w_1 \star f) \star w_2 \quad (\mathbf{3}-36).$$

Example:

If we have an image of size MN and kernel of size mn then convolving the kernel over image requires $MNmn$ multiplications and additions. For each pixel in an image, we need to compute mn multiplications and additions. If kernel has size $3x3$ then for each image pixel, we need to compute 9 multiplications and additions. If image has size MN then this count will be $MNmn$.

If kernels are separable then a 2D matrix of size mn is converted into two 1D matrices of size $1xm$ and $nx1$. Following the associative property of convolution, we can convolve the image in any order with kernel. By convolving with first kernel, we have to do MNm operations and performing convolution with second kernel requires MNn operations totaling to

$$MNm + MNn = MN(m+n) \ll MNmn \qquad (\mathbf{3}-37)$$

<Programming Exercise: Take a separable kernel and prove equation $\mathbf{3-36}$ visually>

<Programming: Write a python program to show number of operations when convolved with combined filter $(G(x,y))$ and when it is convolved with partial filters $(G_1(x), G_2(y))$

Process of Constructing the Spatial Domain Kernels

There are multiple ways to construct a filter

1. Kernel construction using mathematical properties.
   - $Integration$: A filter that works on neighborhood of a pixels and finds its value based on total effect of other pixels i.e. average filter, blurs the image.
   - $Differentiation$: A filter that finds the local difference of a pixel from its neighborhood, can be used to sharpen the image.
2. Considering the shape of 2D function
   - A Gaussian function can be used to weighted average filter
3. Using frequency response
   - We will discuss this type of filter in next chapter

The Smoothing (Low pass) filters

Smoothing or averaging filters are used to reduce the sharp changes in an image. The value of smoothing is determined by two things, size of kernel and value of its coefficients. For example, 5x5 kernel produces higher blur as compared to 3x3 filter. Similarly, a filter A will have more smoothing effect than filter B

$$A = \begin{bmatrix} 2 & 4 & 2 \\ 4 & 8 & 4 \\ 2 & 4 & 2 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad (\mathbf{3}-37)$$

It can be applied in situations:

- $Noise\ reduction$: Noise can introduce sharp changes in image pixel intensities. In order to remove such sharp changes, an averaging filter can be used.
- $Reducing\ irrelevant\ details$: Smoothing can help in reducing unwanted details in given image. Hence it can be used in application where we want to prominent feature of given image. Usually details in a region that is smaller than kernel size is dumped.
- $False\ contour\ correction$: If insufficient intensity levels are used in an image, they introduce false lines, smoothing can be used to remove such false contours from image.
- $Image\ enhancement$: smoothing filter can be used in combination with other techniques to enhance the quality of an image.

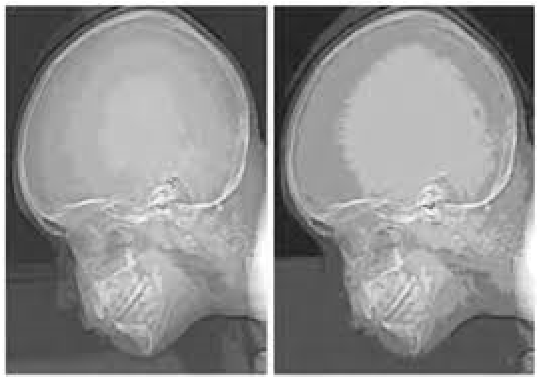


Figure Error! Use the Home tab to apply 0 to the text that you want to appear here.**-5: A) Good quality image B) Image with false contours**

Box filter kernels

It is simple, separable, low pass filter kernel whose coefficients have same value, usually 1 as shown below:

$$box\ filter\ kernel = \frac{1}{C}\begin{bmatrix}1 & 1 & 1\\1 & 1 & 1\\1 & 1 & 1\end{bmatrix} \text{ where } 1 \le C \qquad (\mathbf{3} - 38)$$

Here C is some normalizing constant. The constant C plays two roles:

- The average value of an area of constant intensity would equal to the intensity in the filtered image
- It prevents introducing a bias during filtering. The sum of the pixels in the original and filtered images will be the same

Box filters are good for quick experimentation as they are simple and efficient to apply. There are few limitations that are associated with box filter kernels i.e. box filters are poor approximations to the blurring characteristics of lenses and box filters favor blurring along perpendicular directions.

<write a program that applies box filter kernel on read image by using kernel of different sizes and using different constants C. Discuss the results and effect on output image>

Low pass Gaussian Filter Kernels

These filters are also known as $circularly\ symmetric$ or $isotropic$ because their response is independent of their orientation. These have usually shape of Gaussian distribution and given as follows:

$$w(s,t) = G(s,t) = Ke^{\frac{-(s^2+t^2)}{2\sigma^2}} = Ke^{\frac{-r}{2\sigma^2}} \qquad (\mathbf{3} - 39)$$

It is observed that small Gaussian kernels have little effect on output image. Useful size of Gaussian kernel should be of size $6\sigma\ x\ 6\sigma$.

Usually Gaussian filter produces more uniform smoothing than box filter kernels.

Non-Linear Filters

Non-linear spatial filters whose response is based on ordering (ranking) the pixels contained in the image region covered by the filter. Smoothing is achieved by replacing the value of the center pixel with the value determined by the ranking result.

Mean Filter

Median filters provide excellent noise reduction capabilities for certain types of random noise with less blurring as compared to box and gaussian filters. These filters are effective when salt-pepper noise (impulse noise) is to be reduced from image.

A median is the center value of a sequence of values when they are arranged. If there is no single center value then two center values are summed and average is used as median. For example, in following sequence A

$$A = \{10, 15, 20, 20, \mathbf{20}, 20, 20, 25, 100\}$$
$$B = \{10, 15, 20, 20, \mathbf{20}, \mathbf{22}, 25, 26, 50, 100\}$$

20 is the center value but in case of sequence B, median value is $(20 + 22)/2 = 21$. Median filter works as follows:

1. A window of size mxn is defined to scroll over the image
2. Image is padded with zeros by adding rows of zero values and columns of zero values in such a way that when window center element is placed on first pixel of picture, all elements of filter window match with either zero values or non-zero values.
3. Median filter is placed on first element of original image.

4. All image pixel values covered by median window are arranged in an order and center element of this arranged array is taken as output value for that pixel in output image.
5. Median window is slides to right by one column. And perform the step 4.
6. If the window has reached to last element in image row, slide it back to first column and also slide window down by one row and continue the process.

Example:

Let we consider the following image and 3x3 window for median filter:

$$f=\begin{bmatrix} 1 & 2 & 0 & 1 & 2 \\ 2 & 3 & 1 & 1 & 2 \\ 1 & 4 & 2 & 2 & 0 \\ 3 & 2 & 3 & 3 & 0 \\ 1 & 0 & 0 & 2 & 1 \end{bmatrix} \qquad (\mathbf{3}-40)$$

We list the picture elements covered by rectangle in ordered form $\{0,1,1,1,\mathbf{2},2,2,3,4\}$. Here 2 is our median. So, every pixel value in output image is replaced by median value of its corresponding window. Following figure shows few steps in this process:

$$\underline{\textit{Zero Padded Input Image}}\ (\mathbf{3}-41)$$

$$\begin{bmatrix} 0&0&0&0&0&0&0\\ 0&1&2&0&1&2&0\\ 0&2&3&1&1&2&0\\ 0&1&4&2&2&0&0\\ 0&3&2&3&3&0&0\\ 0&1&0&0&2&1&0\\ 0&0&0&0&0&0&0 \end{bmatrix} \rightarrow \begin{bmatrix} 0&0&0&0&0&0&0\\ 0&1&2&0&1&2&0\\ 0&2&3&1&1&2&0\\ 0&1&4&2&2&0&0\\ 0&3&2&3&3&0&0\\ 0&1&0&0&2&1&0\\ 0&0&0&0&0&0&0 \end{bmatrix} \rightarrow \begin{bmatrix} 0&0&0&0&0&0&0\\ 0&1&2&0&1&2&0\\ 0&2&3&1&1&2&0\\ 0&1&4&2&2&0&0\\ 0&3&2&3&3&0&0\\ 0&1&0&0&2&1&0\\ 0&0&0&0&0&0&0 \end{bmatrix} \rightarrow \begin{bmatrix} 0&0&0&0&0&0&0\\ 0&1&2&0&1&2&0\\ 0&2&3&1&1&2&0\\ 0&1&4&2&2&0&0\\ 0&3&2&3&3&0&0\\ 0&1&0&0&2&1&0\\ 0&0&0&0&0&0&0 \end{bmatrix}$$

$$\begin{bmatrix} 0&1&1&1&x\\ x&x&x&x&x\\ x&x&x&x&x\\ x&x&x&x&x\\ x&x&x&x&x \end{bmatrix}$$

$$\underline{\textit{Median filtered output image}}$$

<fill the rest of the matrix>

<write a program to find the mean filter of an image>

Min and Max Filter

Just like median filter, where we select center value in ordered list, in min-filter, we select minimum value from that ordered list as value of output pixel. And for max-filter, maximum value is selected. As an example, in right most matrix in equation $(\mathbf{3}-40)$ value 0 will be output if min-filter is used and 2 will be outout if max-filter is used.

<find the min and max filtered output using input matrix of equation $(\mathbf{3}-40)$>

<write python program to apply min,max and median filter on input image and discuss your results>

The Sharpening (High pass) filters

This filtering process high lights sharp changes in intensity. It is useful in applications where user need to detect such changes in an image. For example, change of tissue intensity in brain MRI. There are numerous applications of sharpening in medical, industry, auto guided vehicles, agriculture, food quality measurement and military and this list is not exhaustive. Smoothing is an aggregate process that lowers the sharpness of an image. Sharpening is the inverse of aggregate and if smoothing is analogous to integration then differentiation is analogous to differentiation.

$$\text{"}A\ differentiation\ is\ the\ difference\ in\ dependent\ variable\ with\ respect\ to\ independent\ variable\text{"}$$

Differentiation and Its properties

Before discussing the sharpening techniques, let we have an overview of differentiation process and its properties. We define differentiation as:

$$if \qquad y = f(x) \qquad \text{then}$$

$$y' = \frac{dy}{dx} = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \qquad (\mathbf{3} - 41)$$

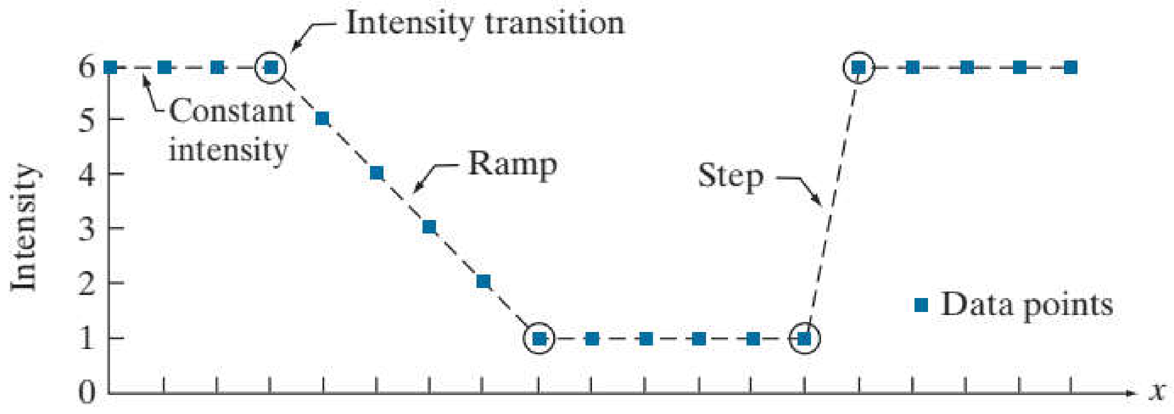
In case of digital image, the difference between two pixels is discrete. If we intend to find difference between two consecutive pixels, the distance (Δx) will be 1 and our equation can be simplified as:

$$\frac{dy}{dx} = f(x+1) - f(x) \qquad (\mathbf{3} - 42)$$

This is called first order derivative as we find difference between two (consecutive) points.

$\boldsymbol{First\ \ order\ derivatives}$ have following properties:

1. Must be zero in areas of constant intensity i.e. there is zero difference between two pixels if they have same intensity value.
2. Must be non-zero on the onset of intensity step i.e. intensity value changes between two (consecutive) pixels.
3. Must be non-zero on intensity ramp i.e. if each pixel value changes along a line, there will be non-difference between two pixels



Values of scan line	6	6	6	6	5	4	3	2	1	1	1	1	1	1	6	6	6	6	6
1st derivative		0	0	−1	−1	−1	−1	−1	0	0	0	0	0	5	0	0	0	0	

Second order derivatives have following properties:

1. Must be zero in areas of constant intensity
2. Must be nonzero at the onset and end of an intensity step or ramp
3. Must be zero along intensity ramps (where intensity changes at same rate)

The second order derivative (double difference) can be expressed mathematically as:

$$y'' = \frac{d^2y}{dx} = \frac{dy}{dx} - \frac{dy}{dx} = [f(x+1) - f(x)] - [f(x) - f(x-1)]$$

$$= f(x+1) - 2f(x) + f(x-1) \qquad (3-43)$$

For double difference, there must be three points i.e. $(x+1), (x), (x-1)$.

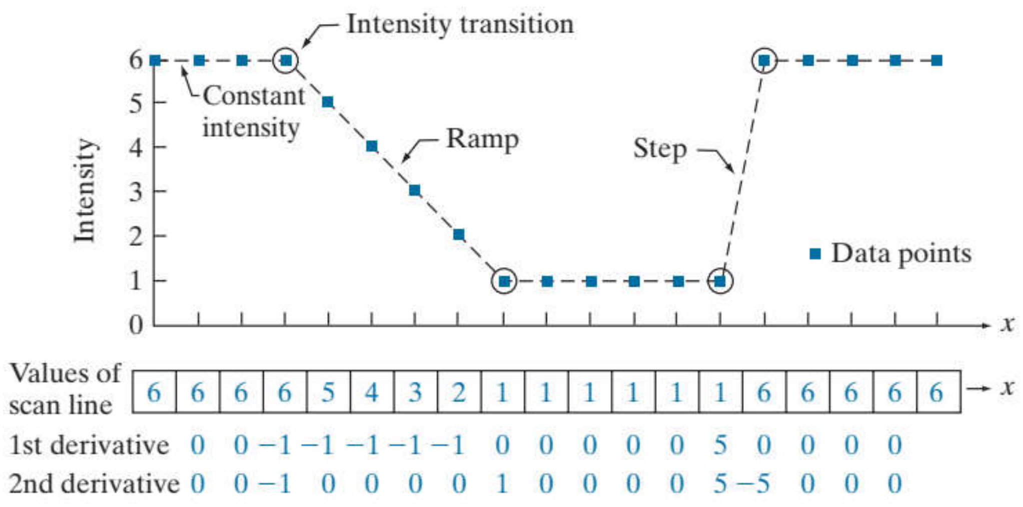
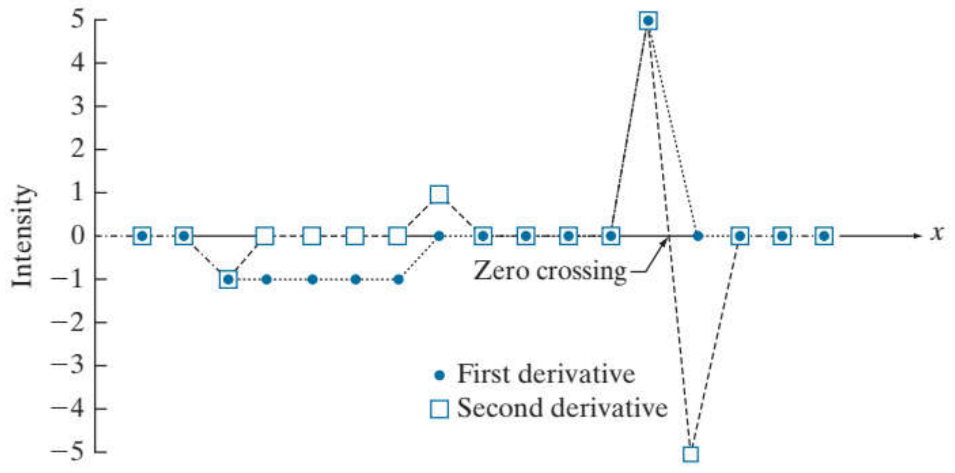


Figure Error! Use the Home tab to apply 0 to the text that you want to appear here.**-6: Derivatives in different image regions**



This $zero\ crossing$ property is quite useful for locating edges.

In digital images, edges are often form the ramp profile in which when an intensity change starts, it remains constant change till it ends as shown in figure 3-5. In this type of intensity profile, first order derivative exists from start of ramp to its end. All the ramp area forms the image and a thicker image is obtained.

On the other hand, in case of second order derivative, 2nd order derivative exists on start of ramp and end of ramp. In between these two points, due to constant intensity change, 2nd order derivative is zero and black line is formed. As a result, double edge is formed. Both of these edges will be one pixel thick (thinner than that of first order derivative edge). Hence double order difference is useful for

- Enhancing fine details in an image
- Second order derivative requires fewer operations that first order derivatives i.e. efficient computationally.

If we find first order difference or 2nd order difference in an image, the result is again a matrix of values known as first order difference image and second order difference image.

<write a python program to compute first order difference image>

<Write a python program to compute second order difference image>

First derivative for image sharpening – The Gradient

First derivatives in image processing are implemented using the magnitude of the gradient. The $\boldsymbol{gradient}$ of an image f at coordinates (x, y) is defined as the two-dimensional column vector:

$$\nabla f = grad(f) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\sigma f}{\sigma x} \\ \frac{\sigma f}{\sigma y} \end{bmatrix} \qquad (\mathbf{3}-44)$$

Equation $(\mathbf{3}-44)$ has two important properties:

- It points in the direction of the greatest rate of change of f at location (x, y)
- The magnitude of vector ∇f is the value of rate of change in the direction of gradient vector. As magnitude is matrix of differences in both x and y direction then it also forms an image called $gradient\ image$.

$$M(x,y) = \|\nabla f\| = mag(\nabla f) = \sqrt{g_x^2 + g_y^2} \qquad (\mathbf{3}-45)$$

The magnitude can also be estimated using following formula:

$$M(x,y) = \|\nabla f\| = mag(\nabla f) = |g_x| + \ |g_y| \qquad (\mathbf{3}-46)$$

- As the components of ∇f are derivatives (differences), they are linear operations however its magnitude is not linear as it incorporates square and square root.
- Partial derivatives are not rotation invariant however $M(x, y)$ is rotation invariant.

Using equation (3-46), we can define g_x and g_y as follows (as defined by Roberts [1965]):

$$g_x = f(x+1, y+1) - f(x, y) \qquad (\mathbf{3}-47)$$

$$g_y = f(x, y+1) - f(x+1, y) \qquad (\mathbf{3}-48)$$

We can now easily find the magnitude by plugging the values of equations $(\mathbf{3}-47)\ and \quad (\mathbf{3}-48)$ in equation (3-45). Equations $3-47$ and $3-48$ are also called the $\boldsymbol{Robert's\ cross\ gradient\ operators}$ and in matrix form they are listed as follows:

$\underline{Robert's\ Operators}$

$$A = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \quad (\mathbf{3}-49)$$

we prefer to use kernels of odd sizes because they have a unique. Such smallest kernel is of size 3x3. For a pixel located at some arbitrary location (x, y), the 3x3 filter can be expressed as:

$$w = \begin{bmatrix} f(x-1,y-1) & f(x,y-1) & f(x+1,y-1) \\ f(x-1,y) & f(x,y) & f(x+1,y) \\ f(x-1,y+1) & f(x,y+1) & f(x+1,y+1) \end{bmatrix} \quad (\mathbf{3}-50)$$

Now the gradient vectors are:

$$g_x = \frac{\partial f}{\partial x} = [f(x-1,y+1) + 2f(x,y+1) + f(x+1,y+1)] - [f(x-1,y-1) + 2f(x,y-1) + f(x+1,y-1)] \; (\mathbf{3}-51)$$

And

$$y = \frac{\partial f}{\partial y} = [f(x+1,y-1) + 2f(x+1,y) + f(x+1,y+1)] - [f(x-1,y-1) + 2f(x-1,y) + f(x-1,y+1)] \; (\mathbf{3}-52)$$

These equations are called the $\boldsymbol{Sobel's\ Gradient\ operators}$ and can be implemented as follows:

$$\underline{Sobel's\ Operators}$$

$$A = \begin{bmatrix} -1 & 2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \qquad B = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad (\mathbf{3}-53)$$

It is to note that coefficients in all the kernels are equal to **zero**.

<write a program in python to apply Robert's operators, Sobel Filters and discuss their results in images>

Second derivative for image sharpening – The Laplacian

An isotropic kernel is one that is independent of rotation. Simplest isotropic kernel (derivative operator) is known as Laplacian kernel. For an image $f(x, y)$, the Laplacian kernel is given as:

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x} + \frac{\partial^2 f}{\partial y} \qquad (\mathbf{3}-54)$$

Equation $(\mathbf{3}-54)$ states that Laplacian could be found by taking double difference in an image first along x-axis and the along y-axis. As defined earlier, for 2nd derivative (double difference), three points are needed. In case of 2D image, we can extend our equation $(\mathbf{3}-43)$ as:

$$\frac{\partial^2 f}{\partial x} = f(x+1,y) - 2f(x,y) + f(x-1,y) \qquad (\mathbf{3}-55)$$

And

$$\frac{\partial^2 f}{\partial y} = f(x,y+1) - 2f(x,y) + f(x,y-1) \qquad (\mathbf{3}-56)$$

And finally

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x} + \frac{\partial^2 f}{\partial y} = f(x+1,y) + f(x-1,y) + \ f(x,y+1) + f(x,y-1) - 4f(x,y) \; (\mathbf{3}-57)$$

The Laplacian could easily be converted to a kernel as follows

$$\frac{\partial^2 f}{\partial x} = \begin{bmatrix} 0 & 0 & 0 \\ f(x-1,y) & -2f(x,y) & f(x+1,y) \\ 0 & 0 & 0 \end{bmatrix}$$

$$\frac{\partial^2 f}{\partial y} = \begin{bmatrix} 0 & f(x,y-1) & 0 \\ 0 & -2f(x,y) & 0 \\ 0 & f(x,y+1) & 0 \end{bmatrix}$$

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x} + \frac{\partial^2 f}{\partial y} = \begin{bmatrix} 0 & f(x,y-1) & 0 \\ f(x-1,y) & -4f(x,y) & f(x+1,y) \\ 0 & f(x,y+1) & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad (\mathbf{3}-58)$$

The Laplacian operator can be extended to produce multiple variations as shown in $(\mathbf{3}-59)$.

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad D = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad (\mathbf{3}-59)$$

Laplacian operator emphasizes the sharp changes in the image and it de-emphasizes the slowly changing regions. This will tend to produce images that have grayish edge lines and other discontinuities, all superimposed on a dark, featureless background. Background features can be “recovered” while still preserving the sharpening effect of the Laplacian by adding the Laplacian image to the original. If the definition used has a negative center coefficient, then we subtract the Laplacian image from the original to obtain a sharpened result. Thus, the basic way in which we use the Laplacian for image sharpening is:

$$g(x,y) = f(x,y) + c[\nabla^2 f(x,y)] \quad (\mathbf{3}-60)$$

In equation $(\mathbf{3}-60)$, c will be negative (-1) if center value of kernel is negative otherwise it will be positive (1).

<Write a program in python to apply all four Laplacian kernels to sharpen images>

Unsharp Masking and high boost filtering

Unsharp masking operation is a way to sharpen the image that consists of following three steps:

1. Blur the original image
2. Subtract the blurred image from the original to get the mask
3. Add the weighted mask to original image

Mathematically we can express this algorithm as follows:

$$g_{mask}(x,y) = f(x,y) - f_{blurred}(x,y) \quad (\mathbf{3}-61)$$

$$g(x,y) = f(x,y) + k g_{mask}(x,y) \quad (\mathbf{3}-62)$$

Different values of k affect the result differently

- $K = 1$, standard unsharp masking
- $k \geq 1$: high boost filtering
- $k \leq 1$: it reduces the effect of the sharpening

  <write a python program to see the effect of unsharp masking using different values of k>

Applications of Gradients

- The gradient is used frequently in industrial inspection, either to aid humans in the detection of defects or, what is more common, as a preprocessing step in automated inspection

  <write a program to do the edge enhancement in given image>

Other Filters (High pass, Band pass and Band Reject filters)

Filters are normally designed by considering signals and their frequencies. Every quantity that change by changing time or some other quantity can be considered as a signal. Generally speaking, any relationship of two quantities that can be written like $y = f(x)$ can be taken as signal. Every signal has a characteristic that is called the $frequency$. **Frequency** can be defined as repetition of signal behavior.

"$The\ term\ \boldsymbol{filter}\ is\ taken\ from\ frequency\ of\ signal\ that\ is\ any\ mechanism\ that\ stops\ signal\ of\ some\ frequencies\ and\ allows\ others\ to\ pass$"

Frequencies are normally divided into two classes: low frequencies and high frequencies.

$Low\ frequencies$ can find coarse features in an image whereas $high\ frequencies$ can determine the fine details. There are four main categories of filters:

1. $Low\ pass\ filters$: These filters allow low frequencies to pass and stop high frequencies.
2. $Band\ pass\ filters$: These filters allow a range of frequencies to pass and stops frequencies that lie before and after the specified band.
3. $High\ pass\ filters$: These filters allow high frequencies to pass and stop all lower frequencies
4. $Band\ reject\ filters\ (notch\ filter)$: These filters stop the specified band of frequencies and allow frequencies that lie before and after specified band to pass.

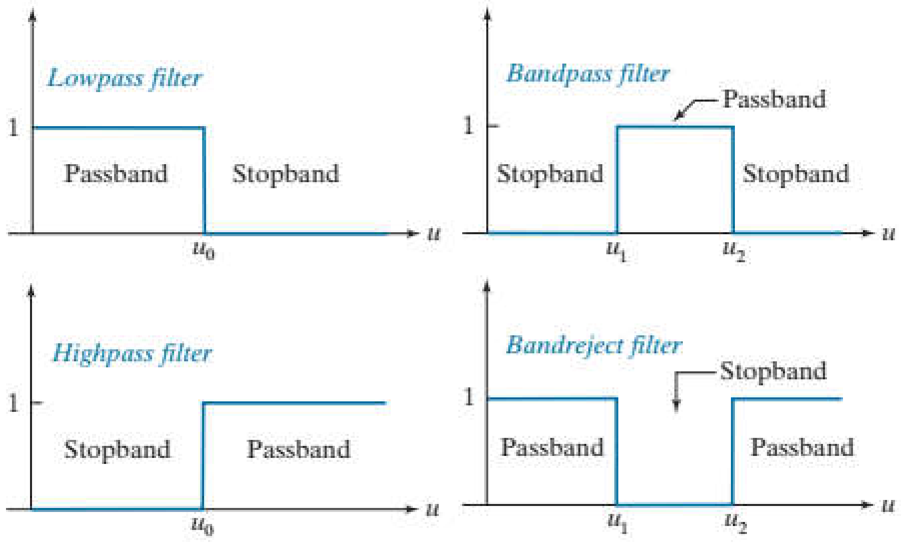


Figure Error! Use the Home tab to apply 0 to the text that you want to appear here.**-7: Visual interpretation of filters**

Low pass filters can be used to construct rest of the three filters. In this section, we will see how can we construct other filters using low pass filters. Following table shows the construction of different filters

Filter Type	Mathematical Description
Low pass filter	$lp(x, y)$
High pass filter	$hp(x, y) = \delta(x, y) - lp(x, y)$
Band reject filter	$br(x, y) = lp_1(x, y) + hp_2(x, y)$
Band pass filter	$bp(x, y) = \delta(x, y) - br(x, y)$

Combining Spatial Enhancement Methods

In order to address a complex image enhancement task, we do combine several image processing methods. As we have studied that:

- Double derivative can be used to find (enhance) fine details in the image

- Single derivative is used to enhance edges i.e. coarser details
- Low pass filters are used to dump fine details

Example

<write a program to apply different image enhancement methods to improve its quality>

Questions

1. write a program to apply log and power law transformation on given images. Also draw the graph between intensity levels and their log and power
2. write a program that applied multi-level thresholding on an input image
3. write a program that extracts each bit plane and displays it
4.