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:

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

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.

Basic Image Transformations

Image Negative

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:

$$s = L - 1 - r \quad (3-2)$$

Example:

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. In python we can calculate the image negative as:

P3-1: Image Negative – ImageNegative.py

```
1   import numpy as np
2   import cv2 as cv
3   img=cv.imread("cameraman.jpg")
4   img1=img[:,:,0]
5   img2=img.copy()
6   L=256
7   rows,cols=img1.shape
8   for x in range(cols):
9       for y in range(rows):
10          r=img1[x,y]
```

11	`s=(L-1)-r`
12	`img2[x,y]=s`
13	`cv.imshow("orignal",img1)`
14	`cv.imshow("negative",img2)`
15	`cv.waitKey()`

Output

Program Description

- *Line* 1, 2: library import
- *Line* 3: Reading an image using *OpenCV*
- *Line* 4: Read image has three channels. This statement extracts only first channel
- *Line* 5: if we assign an object to other object variable, python just copies the references not the original contents. In that case, change made in any variable, will reflect to other variable. In order to create separate copies, we use python "copy()" method.
- *Line* 6: Defining the number of levels
- *Line* 7: Reading image shape
- *Line* 8 − 12: findind the negative of image by subtracting each pixel value from 255
- *Line* 13 − 15: these statements display both original and negative image

Python is very versatile and feature rich language. It supports both Single Instruction and Single Data (SISD) and single instruction multiple data (SIMD) operations. SIMD operations are also called vector operation. For example, we replace lines 8-12 in program P3-1 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 position.

Log Transformation
It is given as

$$s = c log(1 + r) \qquad (3 - 3)$$

Power law (Gamma) Transformation
$$s = cr^{\gamma} \qquad (3 - 4)$$

Piecewise Linear Transformation

It is the process where we apply different type of transformation on different range of intensities.

Piecewise Linear Thresholding

An image contains pixel that have intensity range $0 \rightarrow L - 1$. If decide two threshold t1 and t2 where

- values below t1 are transformed using some function T1,
- pixels from t1 to t2 are transformed using function T2 and
- finally, pixels with values more than t2 are transformed using function T3

The values of t1 and t2 can be either found manually or using some sophisticated algorithm.

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. This can be implemented in several ways. One method could be that intensities of interest should be transformed to 255 and others to 0.
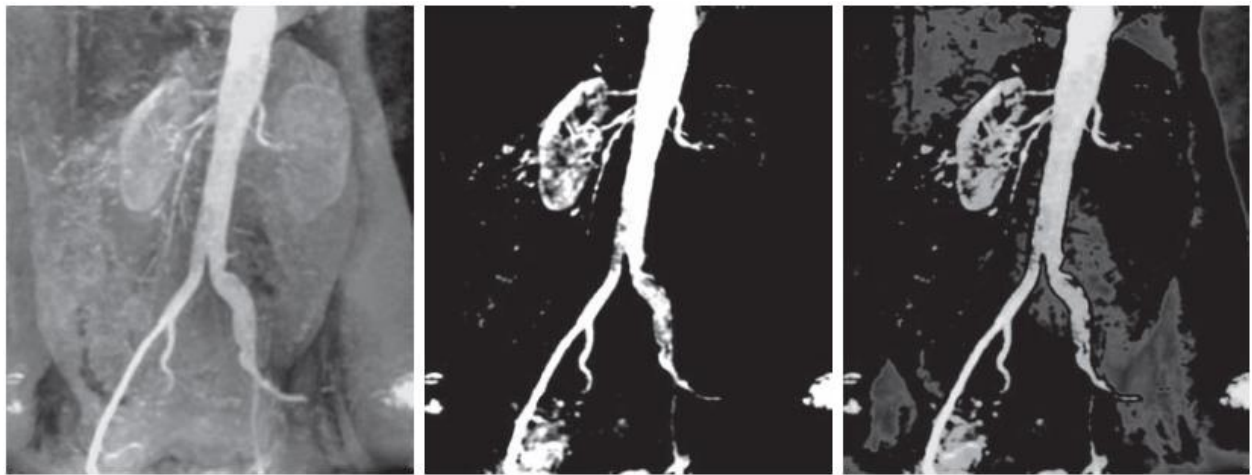
Example

Figure Error! No text of specified style in document.**-1: Application of multi-level intensity thresholding**

Bit-Plane Slicing

Pixel values are integers composed of bits. For example, values in a 256-level grayscale image are composed of 8 bits (one byte). Instead of slicing on intensity levels, we can slice the pixels on their bit levels. I.e. 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:

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:

$$1101\ 1\mathbf{1}10$$

$$\underline{0000\ 0\mathbf{1}00}\ \&$$

$$0000\ 0100$$

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.

Contrast Stretching

We may have images in which majority of pixels lies with in small range of levels and rest of the levels range is either empty of contain very small number of pixels. In such cases, dominant image part is formed by small number of intensity levels and as a result either the image is darker or brighter.

Contrast stretching attempts to distribute pixels over all range of intensity levels and as a result the output image contains more intensity level i.e. more contrast. Most of the time, contrast stretching is done using histogram equivalization that we will discuss later in this chapter.

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.

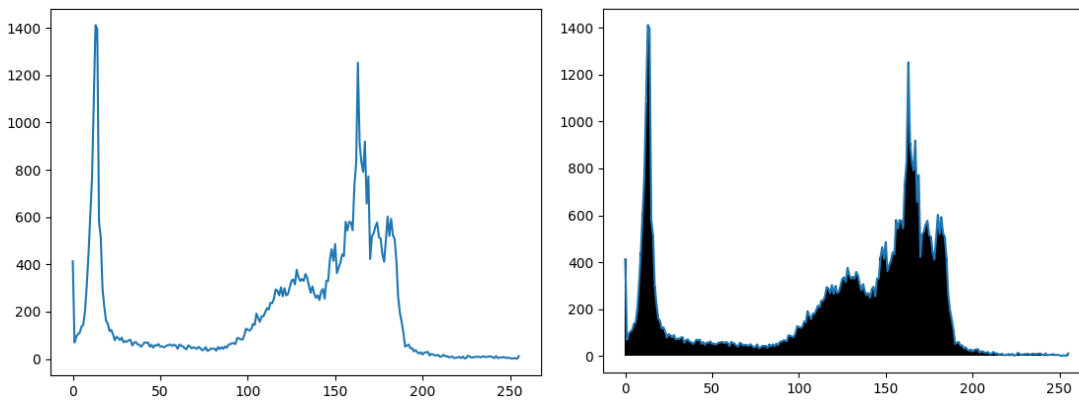| API	Description
x.astype()	This function converts the data type of x as specified in its parameters
plt.plot(arr1,arr2)	This method plots the values in plane. $arr1$ refers to x-axis values and $arr2$ refers to y-axis values.
plt.vlines(x, ymin, ymax)	This method draws vertical line in plot on x location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

```
1   import numpy as np
2   import cv2 as cv
3   import matplotlib.pyplot as plt
```

```
4   img=cv.imread("cameraman.jpg")
5   img1=img[:,:,0]
6   bins=np.zeros(256)
7   for i in range(256):
8         bins[i]=np.sum(img1==i)
9   print(bins.astype(np.uint8))
10  levels=range(256)
11  plt.plot(levels,bins)
12  for i in range(256):
13        plt.vlines(x=i, ymin=0, ymax=bins[i])
14  plt.show()
```

Output

Program Description

- *Line6*: We create an array of 256 elements populated with zeros.
- *Line7*: for loop that iterates for 256 times
- *Line8*: 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.
- *Line9*:   convert the bins datatype to unsigned integer and print its values
- *Line10*: create an array containing elements from 0-255
- *Line11*: plot levels and bins, if we do not create the vertical lines, output plot generated is shown in left figure.
- *Line13*: We create 256 vertical lines to show the bins
- *Line14*: 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

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

P3-3: Image Histogram – ImageHistogram.py

```python
1   import numpy as np
2   import cv2 as cv
3   import matplotlib.pyplot as plt
4   img=cv.imread("cameraman.jpg")
5   img1=img[:,:,0]
6   hist = cv.calcHist(images=[img1],
7   channels=[0],mask=None,histSize=[256],ranges=[0,255])
8   plt.plot(hist, color="black")
9   plt.xlim([0, 256])
10  plt.show()
```

Output
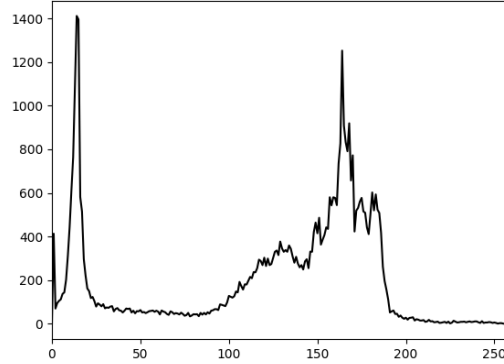
| Program Description |
|---|
| • **Line6,7**: use of OpenCV histogram calculation method.
 • **Line9**: Specifying the range of values on x-axis in plot diagram |

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

$$p_r(k) = \frac{n_k}{MN} \qquad (3-4) \quad \text{and} \quad \sum_k p_r(k) = 1 \qquad (3-5)$$

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:

Histogram of dark image

Histogram of light image

Histogram of low-contrast image

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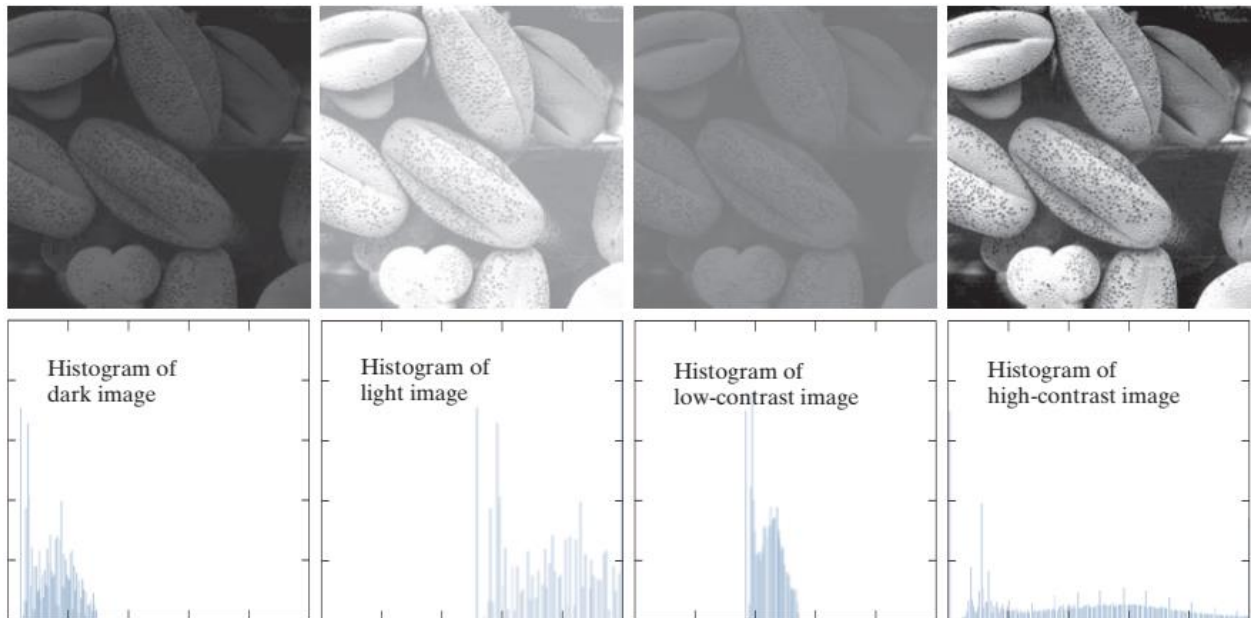
It shows that we should expand the histogram of an image over all available range of intensities in order to get a high-quality image.

<p style="color:red; text-align:center;"><Write a program to shift image histogram to different locations and show the shifted image></p>

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

$$s = T(r) \text{ where } 0 \leq r \leq L-1 \quad (3-6)$$

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.

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

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

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

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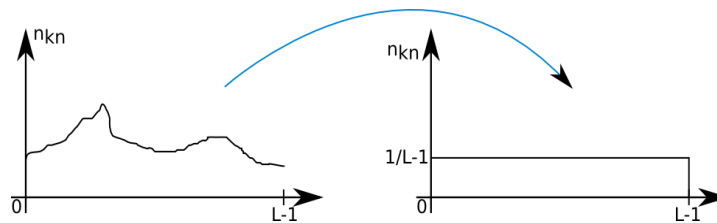
Let for output image, if its CDF=1, it must cover all intensity levels i.e. $L-1$. This can be written mathematically as:

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

(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

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

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

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

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: $Equation\ (3-11)$

Intensity level r_k	Intensity density n_k	Probability density function $p_r(r_k) = \dfrac{n_k}{MN}$	$s_k = T(r_k)$ $= (L-1) \sum_0^k p_i$	$Round(s_k)$	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
$r_4 = 4$	329	0.08	**6.23**	6	=985
$r_5 = 5$	245	0.06	**6.65**	7	245+122
$r_6 = 6$	122	0.03	**6.86**	7	+81
$r_7 = 7$	81	0.02	**7.00**	7	= 448

Now we create histograms of original and equalized images follows:

<Write a program to perform the histogram equalization using python only>
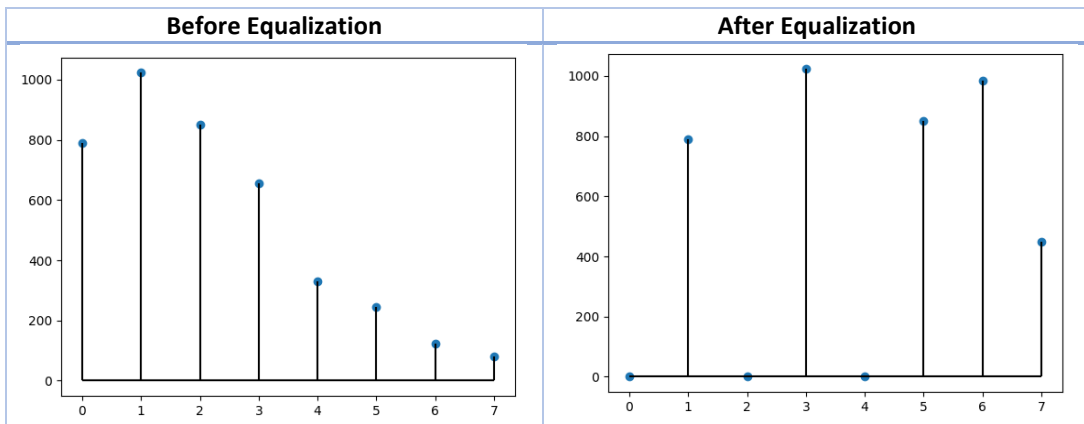
P3-4: Image Histogram – ImageHistogram.py

```
1  import numpy as np
2  import cv2 as cv
3  import matplotlib.pyplot as plt
4  levels=np.array([0,1,2,3,4,5,6,7])
5  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

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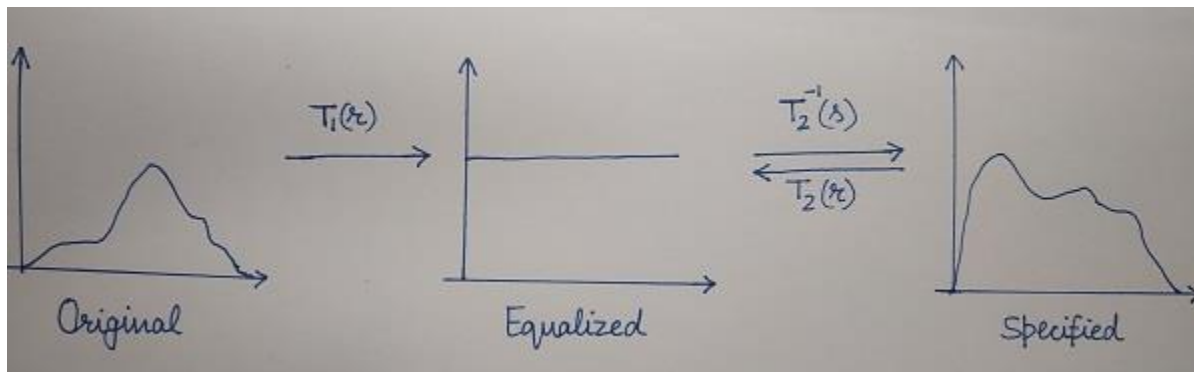
Let s and r are the output and input intensities respectively then transformation is written as:

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

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

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

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

We can write these equations more specifically as:

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

For a value of q that

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

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

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.

Example

Let we have an image with following features:

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

Following tables shows the details of input and reference histograms

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

```
1   import numpy as np
2   import cv2 as cv
3   import matplotlib.pyplot as plt
4
5   r_k=np.array([0,1,2,3,4,5,6,7])
6   n_k=np.array([790,1023,850,656,329,245,122,81])
7   pr_k=n_k/4096
8   s_k=np.zeros(len(n_k))           # for s_k=T(r_k)
```

```
 9    n_s_k=np.zeros(len(n_k))              # pixel mapping n_k --> n_s_k
10    def histEqualization():
11        s_k=np.zeros([len(n_k)])
12        for i in range(len(n_k)):
13            for j in range(0,i+1):
14                s_k[i]+=pr_k[j]
15        s_k=7*s_k
16        s_k=np.round(s_k)
17        for i in range(len(n_k)):
18            n_s_k[i]+=np.sum(n_k[s_k==i])
19        return n_s_k.astype(np.uint8),s_k.astype(np.uint8)
20
21    n_s_k,s_k=histEqualization()
22    pz_k=np.array([0.00,0.00,0.00,0.15,0.20,0.30,0.20,0.15])
23    Gz_q=np.zeros([len(pz_k)])
24    for i in range(len(pz_k)):
25        for j in range(0,i+1):
26            Gz_q[i]+=pz_k[j]
27    Gz_q=7*Gz_q
28    Gz_q=np.round(Gz_q).astype(np.uint8)
29    nz_k=np.zeros(len(n_k))
30    s_k_missing=[]
31    Gz_k_missing=[]
32    for i in range(len(n_k)):
33        ind_gz=Gz_q[i]
34        ind_sk=s_k[i]
35        if s_k.__contains__(ind_gz):
36            val_nk=n_k[s_k==ind_gz]
37            nz_k[i]=sum(val_nk)
38        if not(s_k.__contains__(ind_gz)) and not(Gz_k_missing.__contains__(ind_gz)):
39            Gz_k_missing.append(ind_gz)
40        if not(Gz_q.__contains__(ind_sk))and not(s_k_missing.__contains__(ind_sk)):
41            s_k.__contains__(ind_gz): (ind_sk)
42
43    s_k_missing=np.asarray(s_k_missing).astype(np.uint8)
44    for i in range(len(s_k_missing)):
45        forward=0
46        backward=0
47        for j in range(np.round(len(Gz_q)/2).astype(np.uint8)):
48            forward=s_k_missing[i]+j
49            backward=s_k_missing[i]-j
50            if(Gz_q[forward]+j==s_k_missing[i]):
51                nz_k[forward]+=n_k[np.where(s_k==s_k_missing)]
52                break
53            elif (Gz_q[backward]-j==s_k_missing[i]):
54                nz_k[backward]+=n_k[np.where(s_k==s_k_missing)]
55                break
56    pz_z=nz_k/4096
57    np.set_printoptions(precision=2)
58    print("r_k::",r_k)
59    print("n_k::",n_k)
60    print("n_k::",pr_k)
61    print("s_k::",s_k)
62    print("n_s_k::",n_s_k)
63    print("pz_k::",pz_k)
64    print("Gz_q::",Gz_q)
65    print("nz_k::",nz_k)
66    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"…>

Local Histogram Processing (Local Histogram Equalization-LHE)

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

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

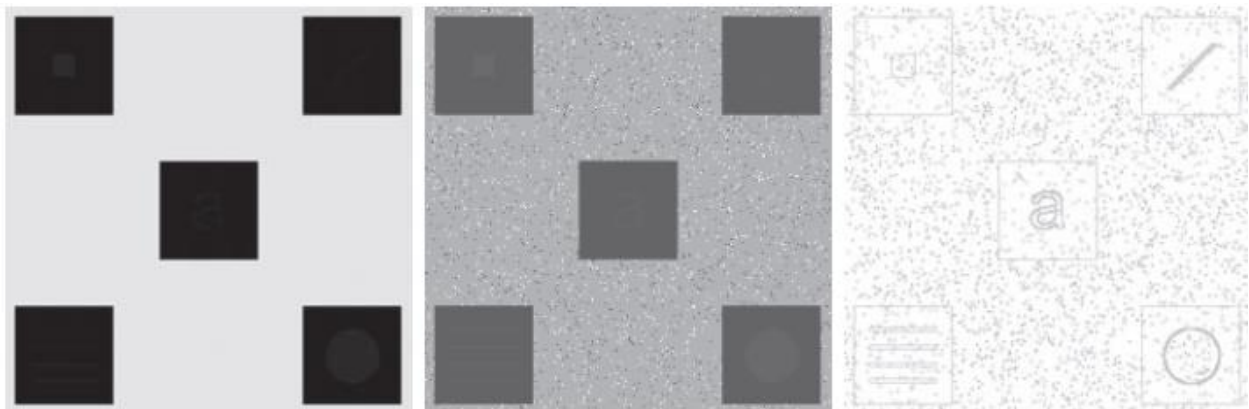Sometimes, it is required to process parts of image in order to extract the hidden information.

Figure Error! No text of specified style in document.**-2: 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.

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.*
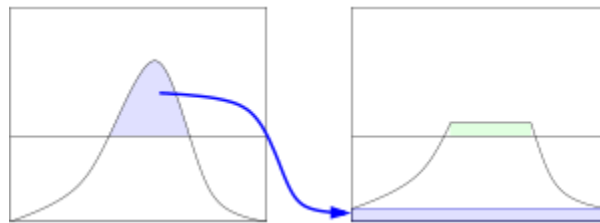

Figure Error! No text of specified style in document.**-3: Contrast bin clipping**

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

```
1   import cv2 as cv
2   img=cv.imread("cameraman.jpg",0)
3   clahe = import numpy as np
4   import cv2 as cv
5   import matplotlib.pyplot as plt
6   import numpy as np
7
8
9
10  img=cv.imread("cameraman.jpg",0)
11  clahe = cv.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
12
```

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

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	$m = \sum_{0}^{L-1} r_i p(r_i) \quad (3-19)$
First moment	$\mu_1 = \sum_{0}^{L-1} (r_i - m)p(r_i) \quad (3-20)$
Second Moment/ Variance σ^2	$\mu_2 = \sum_{0}^{L-1} (r_i - m)^2 p(r_i) \quad (3-21)$
Nth moment	$\mu_n = \sum_{0}^{L-1} (r_i - m)^n p(r_i) \quad (3-22)$

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 σ^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:

$$m_{S_{xy}} = \sum_{0}^{L-1} r_i p_{S_{xy}}(r_i) \quad (3-23)$$

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

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

Example

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 $m, m_{S_{xy}}, \sigma^2, \sigma^2_{S_{xy}}$
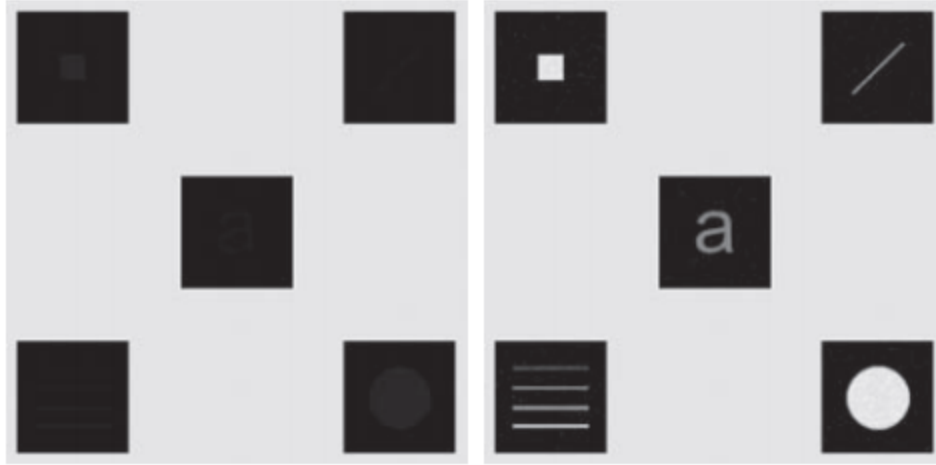
- A pixel $p(x, y)$ is candidate for processing if $k_0 m \leq m_{s_{xy}} \leq k_1 m$ for some non-negative value of k where $k_0 \leq k\_1$. We choose k arbitrary. For example, if our focus is on areas that are darker than one-quarter of the mean intensity we would choose k0 = 0 and k1 = 0.25.
- A pixel as candidate for processing if $k_2 \sigma \leq \sigma_{s_{xy}} \leq k_3 \sigma$. 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

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

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

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

- *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:

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

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 ($w \,\overset{\star}{\star}\, f$)	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

$w \star f$ = [34 22 18 19 23 17 14 9]	$w \star f$ = [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:

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

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

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

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

$= (0*1 + 0*1 + 0*2) + (1*2 + 2*1 + 0*1) + (2*1 + 3*1 + 1*1) = 0 + 4 + 6$
$= $ **10**

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

$= (0*1 + 0*1 + 0*2) + (2*2 + 0*1 + 1*1) + (3*1 + 1*1 + 1*1) = 0 + 5 + 5$
$= $ **10**

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

$= (0*1 + 0*1 + 0*2) + (0*2 + 1*1 + 2*1) + (1*1 + 1*1 + 2*1) = 0 + 5 + 4$
$= $ **9**

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

$= (0*1 + 0*1 + 0*2) + (1*2 + 2*1 + 0*1) + (1*1 + 2*1 + 0*1) = 0 + 4 + 3$
$= $ **7**

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

$= (0*1 + 1*1 + 2*2) + (0*2 + 2*1 + 3*1) + (0*1 + 1*1 + 4*1) = 5 + 5 + 5$
$= $ **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>

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
1   import numpy as np
2   import cv2 as cv
3   import matplotlib.pyplot as plt
4
5   img=cv.imread("cameraman.jpg",0)
6   kernel = np.ones((5,5),np.float32)/25
7   dst = cv.filter2D(img,-1,kernel)
8   plt.subplot(121),plt.imshow(img),plt.title('Original')
9   plt.xticks([]), plt.yticks([])
10  plt.subplot(122),plt.imshow(dst),plt.title('Averaging')
11  plt.xticks([]), plt.yticks([])
12  plt.show()
```

Output ::

Original	Averaging

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

```
1  mat=np.random.randint(1,10,[5,5])
2  # reversing the matrix
3  rows,cols=mat.shape
4  rotated=np.zeros([5,5],np.uint8)
5  for i in range(rows):
6      rotated[i,:]=mat[-(i+1),:]
7  for j in range(cols):
8      rotated[:,j]=mat[:,-(j+1)]
9  print(mat)
10 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	$f \star (g + h) = (f \star g) + (f \star h)$	$f \star (g + h) = (f \star g) + (f \star h)$

<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:

$$w_n \star \dots w_3 \star w_2 \star (w_1 \star f) \quad (3-31)$$

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

$$\boldsymbol{w} = w_1 \star w_2 \star \dots \star w_n \quad (3-32).$$

and

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

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.

$$G(x,y) = G_1(x)G_2(y) \quad (3-34).$$

Example:

Let we have a kernel as follows:

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

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).

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

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

$$MNm + MNn = MN(m+n) \ll MNmn \quad (3-37)$$

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

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

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! No text of specified style in document.**-4: 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:

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

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.

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:

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

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

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

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.
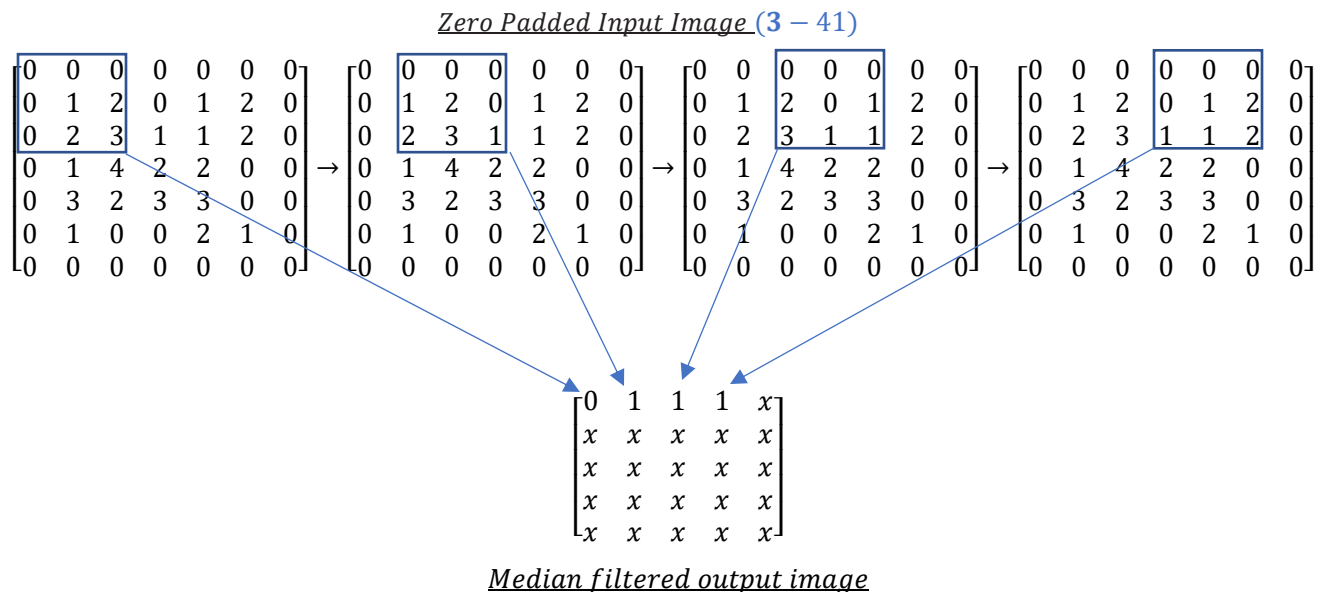3. Median filter is placed on first element of original image.

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:

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

We list the picture elements covered by rectangle in ordered form {0,1,1,1, **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:

Zero Padded Input Image $(3-41)$

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

Median filtered output image

<u>fill the rest of the matrix></u>

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 $(3-40)$ 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 $(3-40)$>

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.

"A differentiation is the difference in dependent variable with respect to independent variable"

Differentiation and Its properties

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

$$if \qquad y = f(x) \qquad then$$

$$y' = \frac{dy}{dx} = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \qquad (3-41)$$
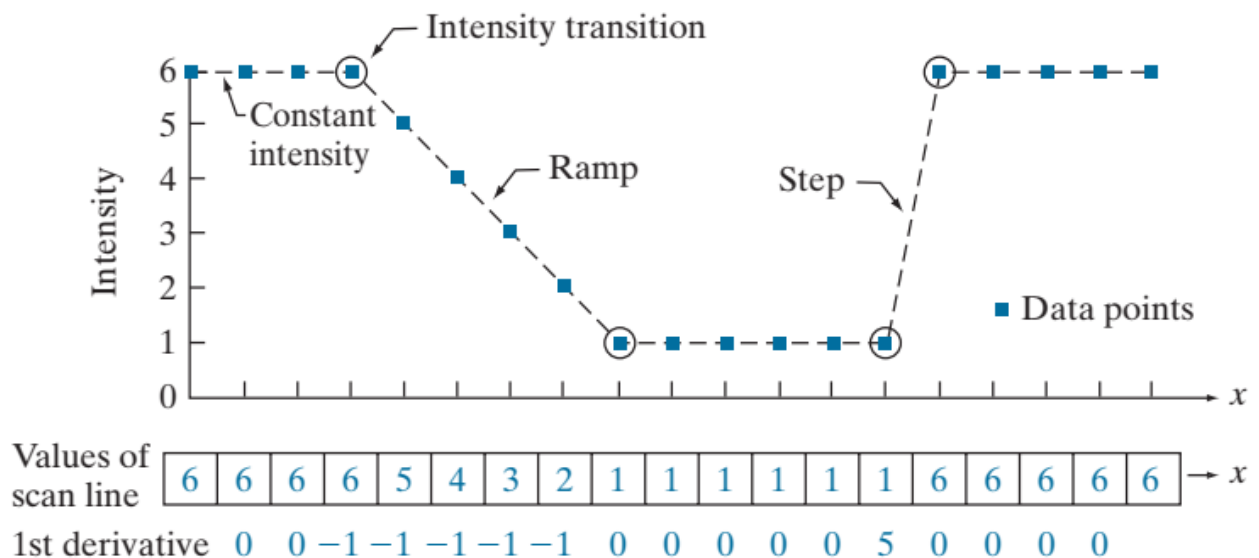
In case of digital image, the difference between two pixels is discrete. If we intend to find difference between two consecutive pixels, the distance (Δx) will be 1 and our equation can be simplified as:

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

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

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	→ x

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:

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

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

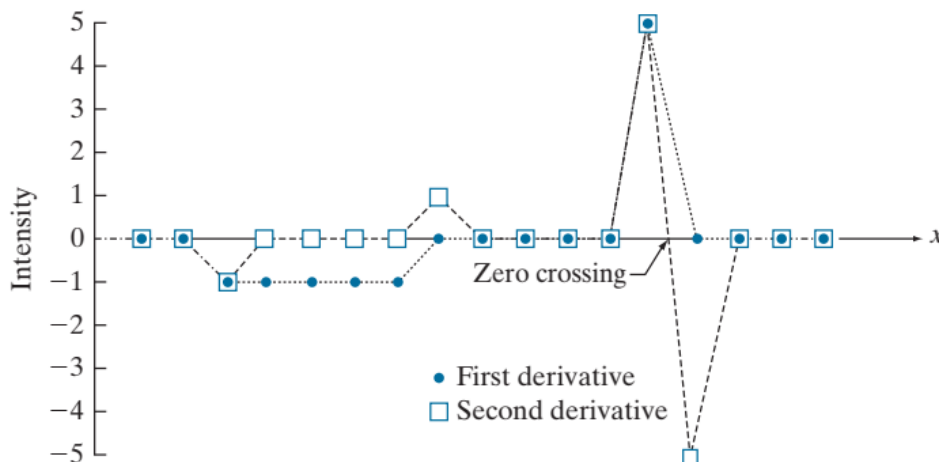For double difference, there must be three points i.e. $(x+1), (x), (x-1)$.
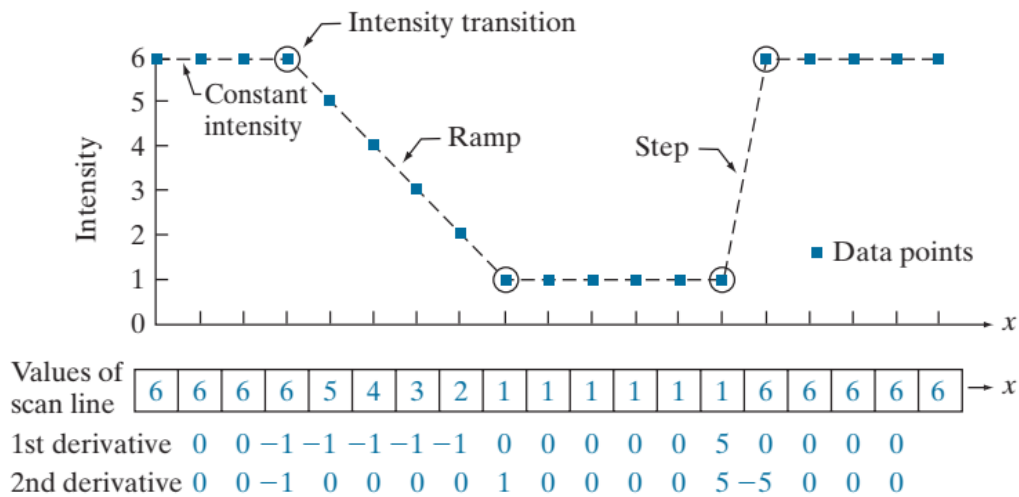
Figure Error! No text of specified style in document.-**5: Derivatives in different image regions**

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 **gradient** of an image f at coordinates (x, y) is defined as the two-dimensional column vector:

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

Equation $(3-44)$ 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*.

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

The magnitude can also be estimated using following formula:
$$M(x, y) = \|\nabla f\| = mag(\nabla f) = |g_x| + |g_y| \qquad (3-46)$$

- 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]):

$$g_x = f(x + 1, y + 1) - f(x, y) \qquad (3-47)$$
$$g_y = f(x, y + 1) - f(x + 1, y) \qquad (3-48)$$

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

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

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

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:

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

Now the gradient vectors are:

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

And

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

These equations are called the **Sobel's Gradient operators** and can be implemented as follows:

Sobel's Operators

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

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

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:

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

Equation $(3-54)$ 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 $(3-43)$ as:

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

And

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

And finally

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

The Laplacian could easily be converted to a kernel as follows

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

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

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

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

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

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:

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

In equation $(3-60)$, 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:

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

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

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

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

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 $filter$ 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.
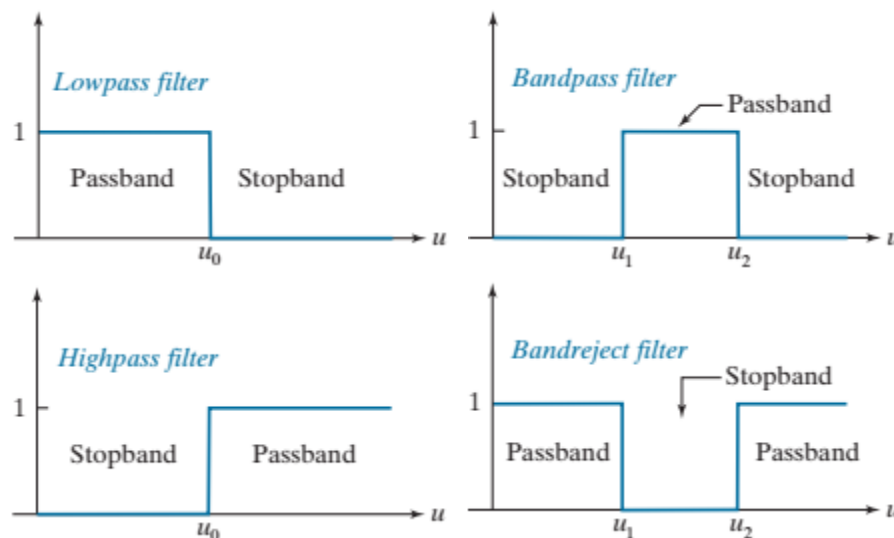

Figure Error! No text of specified style in document.**-6: Visual interpretation of filters**

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>