Image Enhancement Techniques

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

In

Computer Science & Engineering

Under the Supervision of

Ms. Nishtha Ahuja

By

Sonal Thakur

Roll No: 111312

Jaypee University of Information Technology

Waknaghat, Solan – 173234 Himachal Pradesh

CERTIFICATE

This is to certify that project report entitled "***Image Enhancement Techniques***", submitted by *Sonal Thakur* in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science & Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this degree.

Date: May 2015 **Supervisor's Name**

ACKNOWLEDGEMENT

First and formost, I would like to thank my guide Ms. NishthaAhuja for guiding me thoughtfully and efficiently through this project, giving me an opportunity to work at my own pace along my own lines, while providing me with very useful directions whenever necessary as well as for providing necessary information regarding the project & also for their support in completing the project.

I would like to express my gratitude towards my parents & member of the University for their kind co-operation and encouragement which help me in completion of this project.My thanks and appreciations also go to my colleagues in developing the project and people who have willingly helped me out with their abilities.

Date: May2015 Name :Sonal Thakur

TABLE OF CONTENTS

ABSTRACT

The field of Digital Image Processing refers to processing digital images by means of digital computer. One of the main application areas in Digital Image Processing methods is to improve the pictorial information for human interpretation. Most of the digital images contain noise. This can be removed by many enhancement techniques. Filtering is one of the enhancement techniques which is used to remove unwanted information (noise) from the image. It is also used for image sharpening and smoothening.

Some neighborhood operations work with the values of the image pixels in the neighborhood and the corresponding values of a sub image that has the same dimensions as the neighborhood. The sub image is called a “filter”.The aim of this project is to demonstrate the filtering techniques by performing different operations such as smoothening, sharpening, removing the noise etc. This project has been developed using Java language because of its universal acceptance and easy understandability.

INTRODUCTION

Interest in digital image processing methods stems from two principal application areas: improvement of pictorial information for human interpretation; and processing of image data for storage, transformation, and representation for autonomous machine perception.

Image Processing is a technique to enhance raw images received from cameras/sensors placed on satellites, space probes and aircrafts or pictures taken in normal day-to-day life for various applications.
Various techniques have been developed in Image Processing during the last four to five decades. Most of the techniques are developed for enhancing images obtained from unmanned spacecrafts, space probes and military reconnaissance flights. Image Processing systems are becoming popular due to easy availability of powerful personnel computers, large size memory devices, graphics softwares etc.
Image Processing is used in various applications such as:

- Remote Sensing
- Medical Imaging
- Non-destructive Evaluation
- Forensic Studies
- Textiles
- Material Science.
- Military
- Film industry
- Document processing
- Graphic arts
- Printing Industry

METHODS OF IMAGE PROCESSING

There are two methods available in Image Processing: Analog Image Processing and Digital Image Processing

Analog Image Processing refers to the alteration of image through electrical means. The most common example is the television image.
The television signal is a voltage level which varies in amplitude to represent brightness through the image. By electrically varying the signal, the displayed image appearance is altered. The brightness and contrast controls on a TV set serve to adjust the amplitude and reference of the video signal, resulting in the brightening, darkening and alteration of the brightness range of the displayed image.

In *Digital Image Processing* , digital computers are used to process the image. The image will be converted to digital form using a scanner – digitizer [6] (as shown in Figure 1) and then process it. It is defined as the subjecting numerical representations of objects to a series of operations in order to obtain a desired result. It starts with one image and produces a modified version of the same. It is therefore a process that takes an image into another.
The term *digital image processing* generally refers to processing of a two-dimensional picture by a digital computer .In a broader context, it implies digital processing of any two-dimensional data. A digital image is an array of real numbers represented by a finite number of bits.
The principle advantage of Digital Image Processing methods is its versatility, repeatability and the preservation of original data precision.

STEPS IN A TYPICAL IMAGE POSESSING SEQUENCE

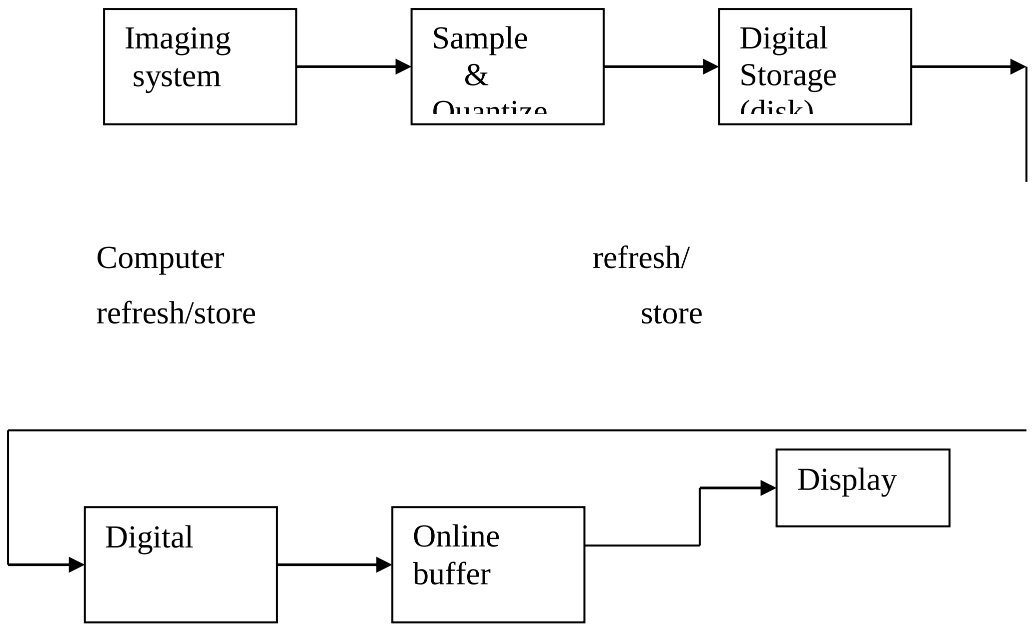


figure 1

An image may be defined as a two-dimensional function, f(x , y), where x and y are spatial coordinates, and the amplitude of f at any pair of coordinates (x , y) is called the intensity or gray level of the image at the point. When x, y, and the amplitude values of f are all finite, discrete quantities, we call the image a digital image. The field of digital image processing refers to processing digital images by means of digital computer.

Digital image is composed of finite number of elements, each of which has a particular location and value. These elements are referred to as picture elements, image elements, pels, and pixels. Pixel is the term most widely used to denote the elements of a digital image. Sometimes a distinction is made by defining image processing as a discipline in which both the input and output of a process are images.

The various Image Processing techniques are:

- Image representation
- Image preprocessing
- ***Image enhancement***
- Image restoration
- Image analysis
- Image reconstruction
- Image data compression

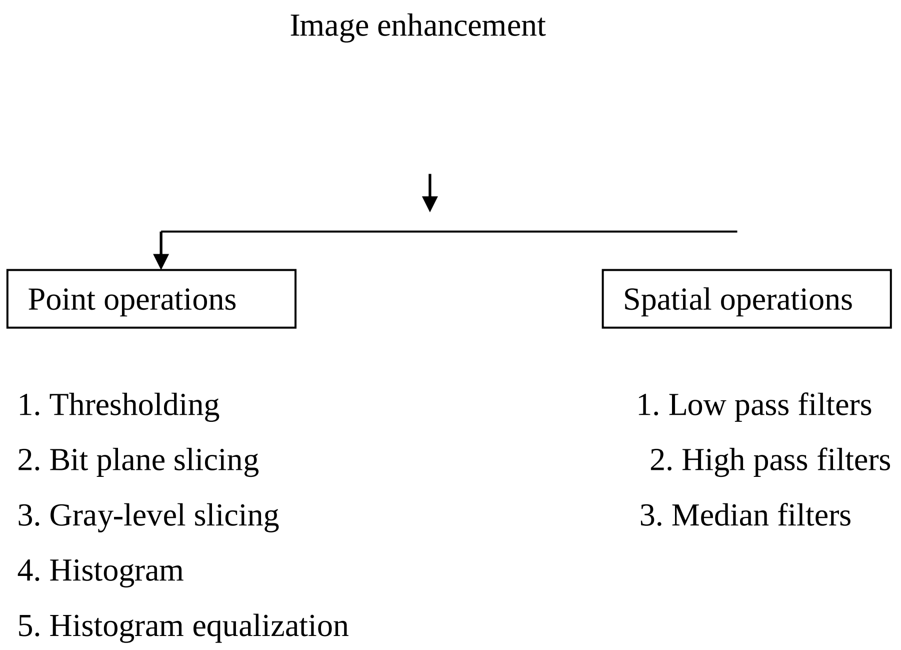


Figure 2

Filters are one of digital image enhancement technique used to sharp t the image and to reduce the noise in the image. There are two types of enhancement techniques called Spatial domain and Frequency domain techniques which are categorized again for smoothing and sharpening the images. The 2D continuous image $a(x,y)$ is divided into *N rows* and *M columns*. The intersection of a row and a column is termed a *pixel*. The value assigned to the integer coordinates $[m,n]$ with $\{m=0,1,2,\ldots,M-1\}$ and $\{n=0,1,2,\ldots,N-1\}$ is $a[m,n]$. In fact, in most cases $a(x,y)$ – which we might consider to be the physical signal that impinges on the face of a 2D sensor – is actually a function of many variables including depth (z), color (λ), and time (t).

FILTERING

- Smoothing filters are used for blurring and for noise reduction.

- Blurring is used in preprocessing steps, such as removal of small details from an image prior to object extraction and bridging of small gaps in lines or curves.

- Noise reduction can be accomplished by blurring with a linear filter and also by linear and also by non linear filtering.

- The principal objective of sharpening is to highlight fine detail in image or enhance detail that has been blurred, either in error or as a natural effect of a particular method of image acquisition.

- Uses of image sharpening vary and include applications ranging from electronic printing and medical imaging to industrial inspection and autonomous guidance in military systems.

Image enhancement approaches fall into two broad categories.

Spatial domain methods:

Frequency domain methods:

SPATIAL DOMAIN METHODS

- Spectral enhancement relies on changing the gray scale representation of pixels to give an image with more contrast for interpretation. It applies the same spectral transformation to all pixels with a given gray scale in an image. However, it does not take full advantage of human recognition capabilities even though it may allow better interpretation of an image by a user.
- Several examples will demostrate the value of spatial characteristics in image interpretation.
- Spatial enhancement is the mathematical processing of image pixel data to emphasize spatial relationships. This process defines homogeneous regions based on linear edges.
- Spatial enhancement techniques use the concept of spatial frequency within an image. Spatial frequency is the manner in which gray-scale values change relative to their neighbors within an image. If there is a slowly varying change in gray scale in an image from one side of the image to the other, the image is said to have a low spatial frequency. If pixel values vary radically for adjacent pixels in an image, the image is said to have a high spatial frequency. Figure 1 (a-b) shows examples of high and low spatial frequencies:

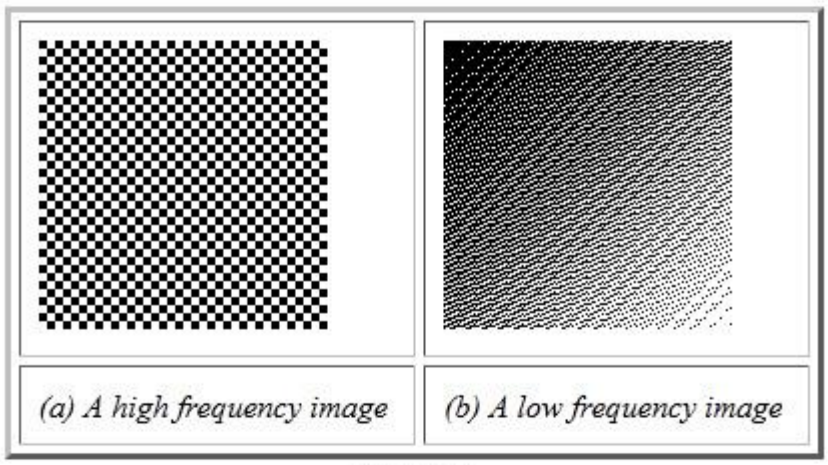


Figure1

- Many natural and manmade features in images have high spatial frequency:
    - Geologic faults
    - Edges of lakes
    - Roads
    - Airports

- Spatial enhancement involves the enhancement of either low or high frequency information within an image. Algorithms that enhance low frequency image information employ a "blurring" filter (commonly called a low pass filter) that emphasizes low frequency parts of an image while de-emphasizing the high frequency components. The enhancement of high frequency information within an image is often called edge enhancement. It emphasizes edges in the image while retaining overall image quality.

Objectives or Purposes There are two main purposes that underlie spatial enhancement techniques:

To improve interpretability of image data
To aid in automated feature extraction

POINT PROCESSING

These techniques are based on gray level mappings, where the type of mapping used depends on the criterion chosen for enhancement. As an eg. consider the problem of enhancing the contrast of an image. Let *r* and *s* denote any gray level in the original and enhanced image respectively. Suppose that for every pixel with level *r* in original image we create a pixel in the enhanced image with level $S = T(r)$. If $T(r)$ has the form as shown

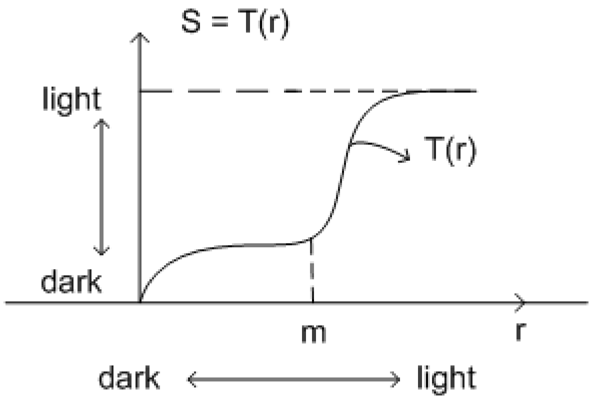


Figure 3

The effect of this transformation will be to produce an image of higher contrast than the original by darkening the levels below a value *m* and brightening the levels above *m* in the original pixel spectrum. The technique is refered to as contrast stretching. The values of *r* below *m* are compressed by the transformation function into a narrow range of *S* towards the dark end of the spectrum; the opposite effect takes place for values of *r* above *m* .

In the limiting case shown in figure, $T(r)$ produces a 2-level (binary) image. This is also referred to as image thresholding. Many powerful enhancement processing techniques can be formulated in the spatial domain of an image.

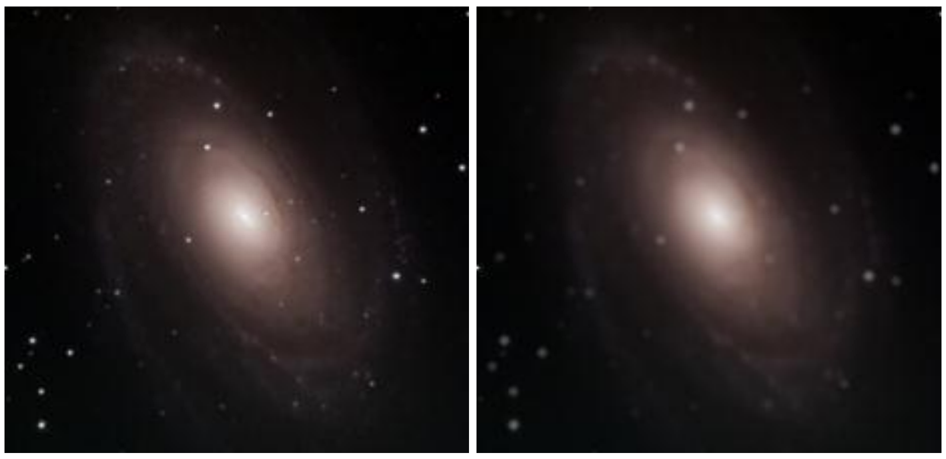
Note: It is to be noted that there is no general theory of image enhancement. When an image is processed for visual interpolation, the observer is the ultimate judge of how well a particular method works. Visual evaluation of image quality is a subjective process thus making the definition of a "good image" an elusive standard by which to compare algorithm performance.

When the problem is one of processing images for machine perception, the evaluation task is easier. For eg. if we take the problem of character recognition by a machine the best image processing method would be the one that yields the best machine recognition result.

TYPES OF SPATIAL OPERATIONS

Low-Pass Filtering (Blurring)

The most basic of filtering operations is called "low-pass". A low-pass filter, also called a "blurring" or "smoothing" filter, averages out rapid changes in intensity. The simplest low-pass filter just calculates the average of a pixel and all of its eight immediate neighbors. The result replaces the original value of the pixel. The process is repeated for every pixel in the image.



Before and After Low-Pass Filter

Figure 4

This low-pass filtered image looks a lot blurrier. But why would you want a blurrier image? Often images can be noisy – no matter how good the camera is, it always adds an amount of ”snow” into the image. The statistical nature of light itself also contributes noise into the image.

Noise always changes rapidly from pixel to pixel because each pixel generates its own independent noise. The image from the telescope isn't "uncorrelated" in this fashion because real images are spread over many pixels. So the low-pass filter affects the noise more than it does the image. By suppressing the noise, gradual changes can be seen that were invisible before. Therefore a low-pass filter can sometimes be used to bring out faint details that were smothered by noise.

MaxIm DL allows you to selectively apply a low-pass filter to a certain brightness range in the image. This allows you to selectively smooth the image background, while leaving the bright areas untouched. This is an excellent compromise because the fainter objects in the background are the noisiest, and it does not degrade the sharpness of bright foreground objects.

Filtering can be visualized by drawing a ”convolution kernel”. A kernel is a small grid showing how a pixel's filtered value depends on its neighbors. To perform a low-pass filter by simply averaging adjacent pixels, the following kernel is used:

+1/9	+1/9	+1/9
+1/9	+1/9	+1/9
+1/9	+1/9	+1/9

When this kernel is applied, each pixel and its eight neighbors are multiplied by 1/9 and added together. The pixel in the middle is replaced by the sum. This is repeated for each pixel in the image.

If we didn't want to filter so harshly, we could change the kernel to reduce the averaging, for example:

0	+1/8	0
+1/8	+1/2	+1/8
0	+1/8	0

The center pixel contributes half of its value to the result, and each of the four pixels above, below, left, and right of the center contribute 1/8 each. This will have a more subtle effect. By choosing different low-pass filters, we can pick the one that has enough noise smoothing, without blurring the image too much.

We could also make the kernel larger. The examples above were 3x3 pixels for a total of nine. We could use 5x5 just as easily, or even more. The only problem with using larger kernels is the number of calculations required becomes very large.

A variation on this technique is a Gaussian Blur, which simply allows you to define a particular shape of blur kernel with just a single number – the radius of a Gaussian ("normal") distribution. This provides a very fine control of the amount of blurring; a larger radius produces a stronger effect.

High-Pass Filtering (Sharpening)

A high-pass filter can be used to make an image appear sharper. These filters emphasize fine details in the image – exactly the opposite of the low-pass filter. High-pass filtering works in exactly the same way as low-pass filtering; it just uses a different convolution kernel. In the example below, notice the minus signs for the adjacent pixels. If there is no change in intensity, nothing happens. But if one pixel is brighter than its immediate neighbors, it gets boosted.

0	-1/4	0
-1/4	+2	-1/4
0	-1/4	0

Unfortunately, while low-pass filtering smooths out noise, high-pass filtering does just the opposite: it *amplifies noise*. You can get away with this if the original image is not too noisy; otherwise the noise will overwhelm the image. MaxIm DL includes a very useful "range-restricted filter" option; you can high-pass filter only the brightest parts of the image, where the signal-to-noise ratio is highest.

High-pass filtering can also cause small, faint details to be greatly exaggerated. An over-processed image will look grainy and unnatural, and point sources will have dark donuts around them. So while high-pass filtering can often improve an image by sharpening detail, overdoing it can actually degrade the image quality significantly.

BLURRING MASKS VS DERIVATIVE MASKS.

We are going to perform a comparison between blurring masks and derivative masks.

BLURRING MASKS:

A blurring mask has the following properties.

- All the values in blurring masks are positive
- The sum of all the values is equal to 1
- The edge content is reduced by using a blurring mask
- As the size of the mask grow, more smoothing effect will take place

DERRIVATIVE MASKS:

A derivative mask has the following properties.

- A derivative mask have positive and as well as negative values
- The sum of all the values in a derivative mask is equal to zero
- The edge content is increased by a derivative mask
- As the size of the mask grows , more edge content is increased

RELATIONSHIP BETWEEN BLURRING MASK AND DERIVATIVE MASK WITH HIGH PASS FILTERS AND LOW PASS FILTERS.

The relationship between blurring mask and derivative mask with a high pass filter and low pass filter can be defined simply as.

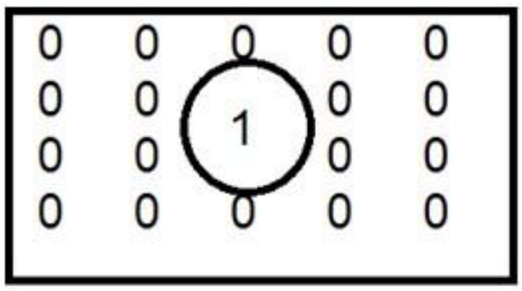
- Blurring masks are also called as low pass filter
- Derivative masks are also called as high pass filter

HIGH PASS FREQUENCY COMPONENTS AND LOW PASS FREQUENCY COMPONENTS

The high pass frequency components denotes edges whereas the low pass frequency components denotes smooth regions.

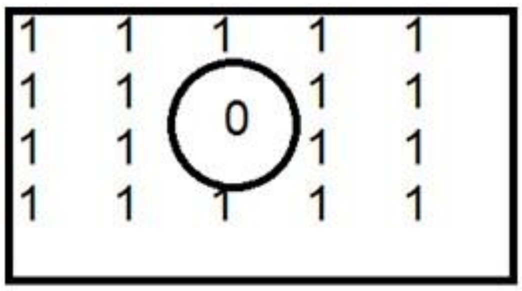
IDEAL LOW PASS AND IDEAL HIGH PASS FILTERS

This is the common example of low pass filter.

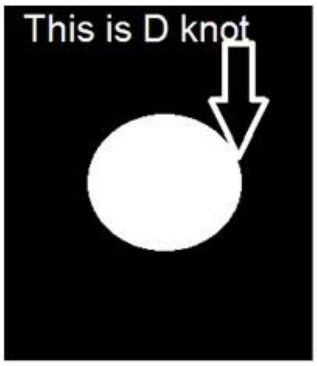


When one is placed inside and the zero is placed outside , we got a blurred image. Now as we increase the size of 1, blurring would be increased and the edge content would be reduced.

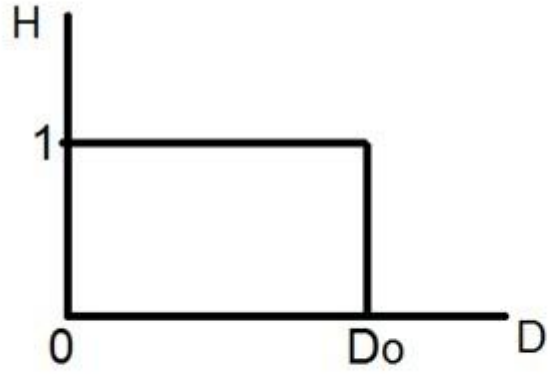
This is a common example of high pass filter.



When 0 is placed inside, we get edges , which gives us a sketched image. An ideal low pass filter in frequency domain is given below



The ideal low pass filter can be graphically represented as



Now let’s apply this filter to an actual image and let’s see what we got.

SAMPLE IMAGE.

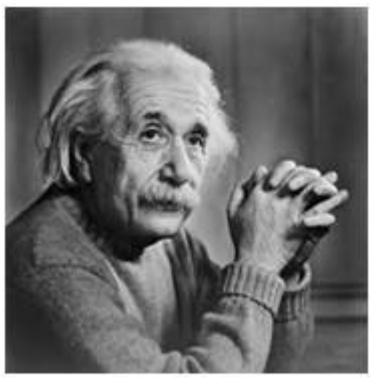
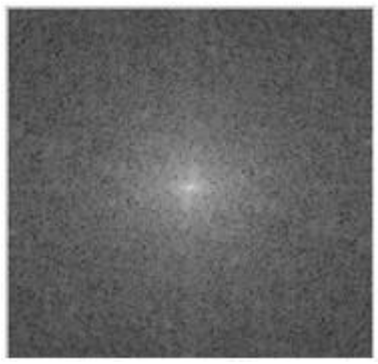
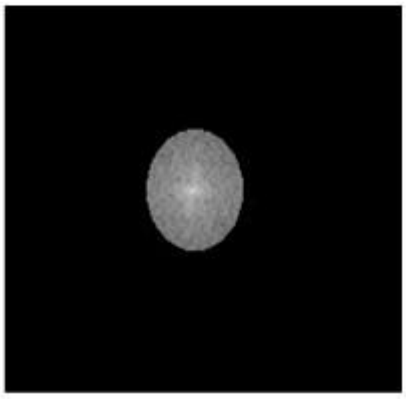


IMAGE IN FREQUENCY DOMAIN



APPLYING FILTER OVER THIS IMAGE



RESULTANT IMAGE

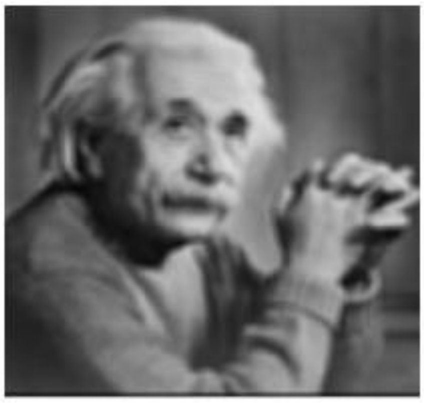


Figure 5

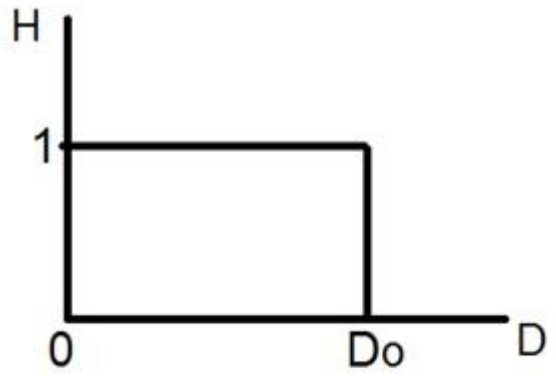
With the same way , an ideal high pass filter can be applied on an image. But obviously the results would be different as , the low pass reduces the edged content and the high pass increase it.

Gaussian Low pass and Gaussian High pass filter

Gaussian low pass and Gaussian high pass filter minimize the problem that occur in ideal low pass and high pass filter.

This problem is known as ringing effect. This is due to reason because at some points transition between one color to the other cannot be defined precisely, due to which the ringing effect appears at that point.

Have a look at this graph.



This is the representation of ideal low pass filter. Now at the exact point of Do , you cannot tell that the value would be 0 or 1. Due to which the ringing effect appears at that point.

So in order to reduce the effect that appears is ideal low pass and ideal high pass filter , the following Gaussian low pass filter and Gaussian high pass filter is introduced.

GAUSSIAN LOW PASS FILTER

The concept of filtering and low pass remains the same, but only the transition becomes different and become more smooth.

The Gaussian low pass filter can be represented as

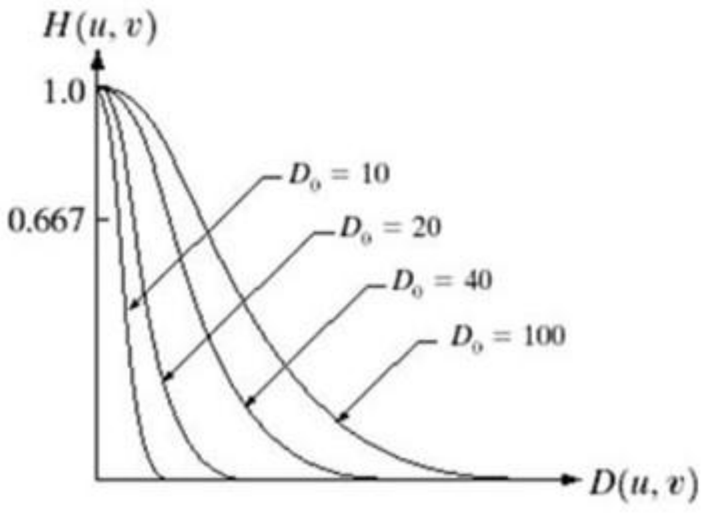


Figure 6

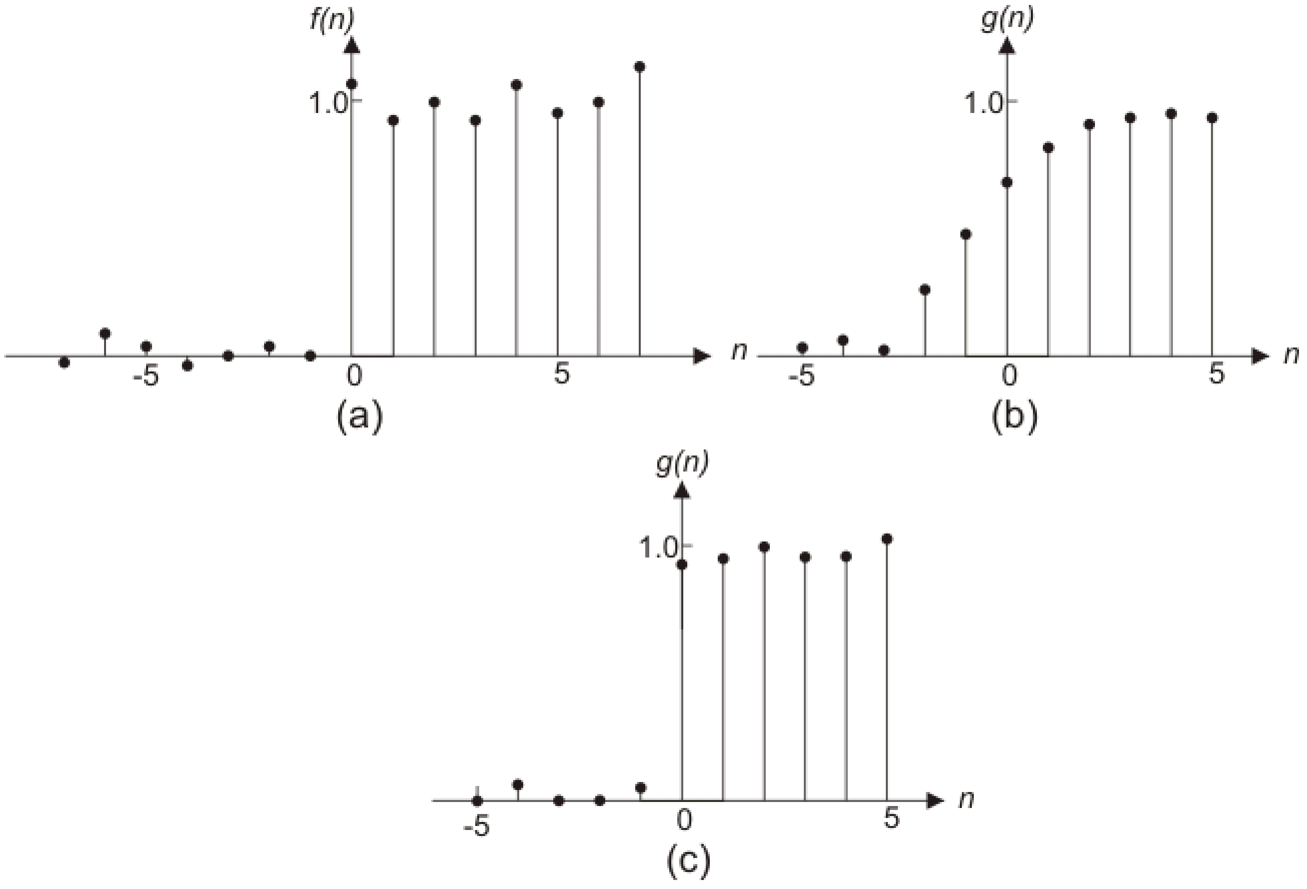
Note the smooth curve transition, due to which at each point, the value of Do , can be exactly defined.

GAUSSIAN HIGH PASS FILTER

Gaussian high pass filter has the same concept as ideal high pass filter , but again the transition is more smooth as compared to the ideal one.

Median Filtering

Median filtering is a nonlinear process useful in reducing impulsive, or salt-and-pepper noise. It is also useful in preserving edges in an image while reducing random noise. Impulsive or salt-and pepper noise can occur due to a random bit error in a communication channel. In a median filter, a window slides along the image, and the median intensity value of the pixels within the window becomes the output intensity of the pixel being processed.For example, suppose the pixel values within a window are 5,6, 55, 10 and 15, and the pixel being processed has a value of 55. The output of the median filter an the current pixel location is 10, which is the median of the five values.



Like lowpass filtering, median filtering smoothes the image and is thus useful in reducing noise. Unlike lowpass filtering, median filtering can preserve discontinuities in a step function and can smooth a few pixels whose values differ significantly from their surroundings without affecting the other pixels.Figure

(5.21) shows a 1-D step sequence degraded by a small amount of random noise. Figure (5.21) shows the result after filtering with a lowpass filter whose impulse response is a 5-point rectangular window. Figure (5.21) shows the result after filtering with 5-point median filter. It is clear from the figure that the step discontinuity is better preserved by the median filter. Figure (5.21a) shows a 1-D sequence with two values that are significantly different from the surrounding points. Figures (b) and (c) show the result of a lowpass filter and a median filter, respectively. The filters used in figure (5.22) are the same as those used in figure(5.21). If the two impulsive values are due to noise, the result of using a median filter will be the reduce the noise. If the two values are part of the signal, however, using the median filter will distort the signal.

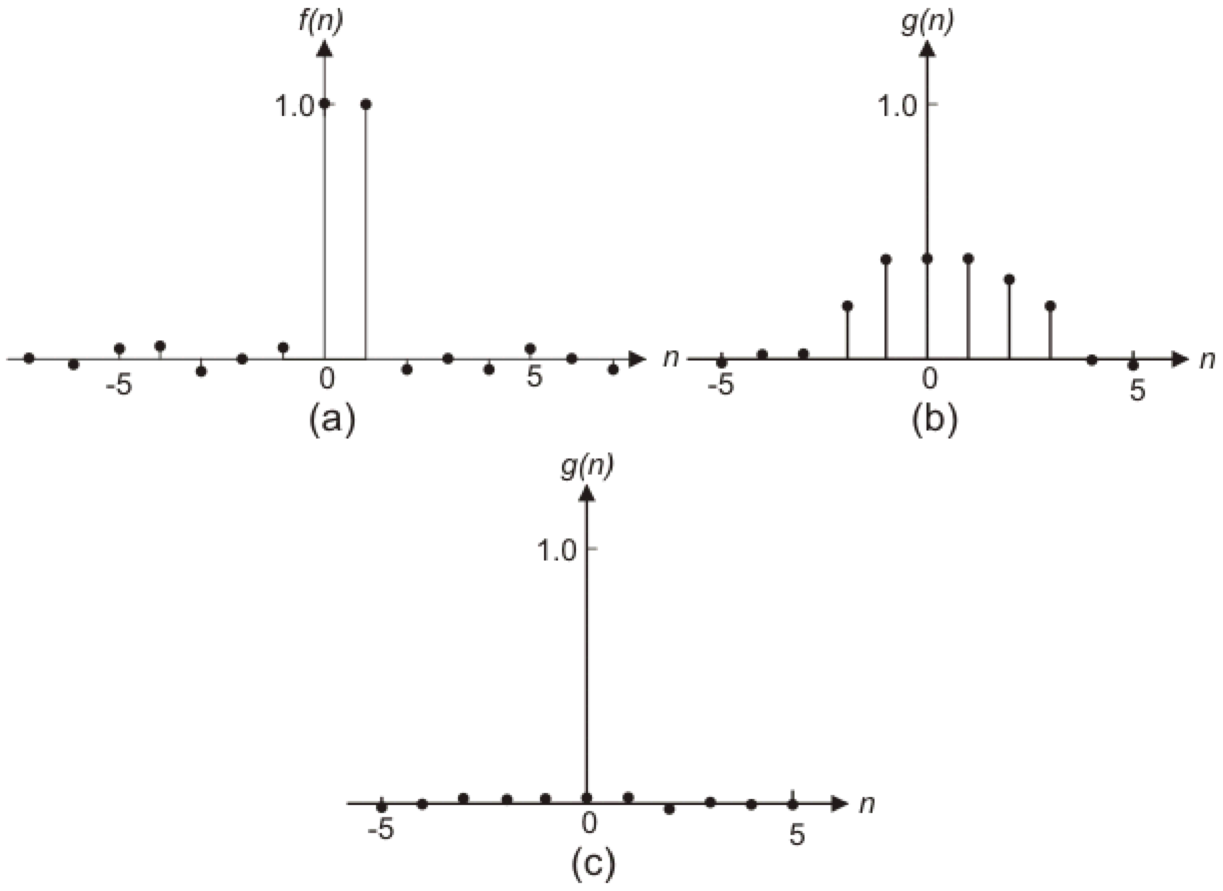


Figure 7

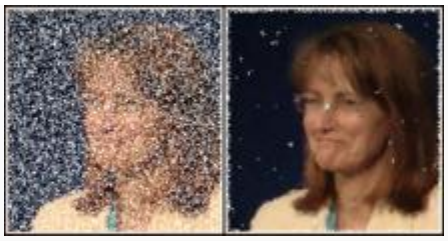
2D median filter pseudo code

Code for a simple 2D median filter algorithm might look like this:

```
allocate outputPixelValue[image width][image height]
allocate window[window width * window height]
edgex := (window width / 2) rounded down
edgey := (window height / 2) rounded down
for x from edgex to image width - edgex
    for y from edgey to image height - edgey
        i = 0
        for fx from 0 to window width
            for fy from 0 to window height
                window[i] := inputPixelValue[x + fx - edgex][y + fy - edgey]
                i := i + 1
        sort entries in window[]
        outputPixelValue[x][y] := window[window width * window height / 2]
```

Note that this algorithm:

- Processes one color channel only,
- Takes the "not processing boundaries" approach (see above discussion about boundary issues).


Use of a median filter to improve an image severely corrupted by defective pixels

Algorithm Implementation Issues

Typically, by far the majority of the computational effort and time is spent on calculating the median of each window. Because the filter must process every entry in the signal, for large signals such as images, the efficiency of this median calculation is a critical factor in determining how fast the algorithm can run. The "vanilla" implementation described above sorts every entry in the window to find the median; however, since only the middle value in a list of numbers is required, selection algorithms can be much more efficient.

Furthermore, some types of signals (very often the case for images) use whole number representations: in these cases, histogram medians can be far more efficient because it is simple to update the histogram from window to window, and finding the median of a histogram is not particularly onerous.

Edge Preserving Properties

Median filtering is one kind of smoothing technique, as is linear Gaussian filtering. All smoothing techniques are effective at removing noise in smooth patches or smooth regions of a signal, but adversely affect edges. Often though, at the same time as reducing the noise in a signal, it is important to preserve the edges. Edges are of critical importance to the visual appearance of images, for example. For small to moderate levels of (Gaussian) noise, the median filter is demonstrably better than Gaussian blur at removing noise whilst preserving edges for a given, fixed window size. However, its performance is not that much better than Gaussian blur for high levels of noise, whereas, for speckle noiseand salt and pepper noise (impulsive noise), it is particularly effective. Because of this, median filtering is very widely used in digital image processing.

TYPES OF POINT OPERATIONS

THRESHOLDING

Brief Description

In many vision applications, it is useful to be able to separate out the regions of the image corresponding to objects in which we are interested, from the regions of the image that correspond to background. Thresholding often provides an easy and convenient way to perform this segmentation on the basis of the different intensities or colors in the foreground and background regions of an image.

In addition, it is often useful to be able to see what areas of an image consist of pixels whose values lie within a specified range, or *band* of intensities (or colors). Thresholding can be used for this as well.

How It Works

The input to a thresholding operation is typically a grayscale or color image. In the simplest implementation, the output is a binary image representing the segmentation. Black pixels correspond to background and white pixels correspond to foreground (or *vice versa*). In simple implementations, the segmentation is determined by a single parameter known as the *intensity threshold*. In a single pass, each pixel in the image is compared with this threshold. If the pixel's intensity is higher than the threshold, the pixel is set to, say, white in the output. If it is less than the threshold, it is set to black.

In more sophisticated implementations, multiple thresholds can be specified, so that a *band* of intensity values can be set to white while everything else is set to black. For color or multi-spectral images, it may be possible to set different thresholds for each color channel, and so select just those pixels within a specified cuboid in RGB space. Another common variant is to set to black all those pixels corresponding to background, but leave foreground pixels at their original color/intensity (as opposed to forcing them to white), so that that information is not lost.

Guidelines for Use

Not all images can be neatly segmented into foreground and background using simple thresholding. Whether or not an image can be correctly segmented this way can be determined by looking at an intensity histogram of the image. We will consider just a grayscale histogram here, but the extension to color is trivial.

If it is possible to separate out the foreground of an image on the basis of pixel intensity, then the intensity of pixels within foreground objects must be distinctly different from the intensity of pixels within the background. In this case, we expect to see a distinct peak in the histogram corresponding to foreground objects such that thresholds can be chosen to isolate this peak accordingly. If such a peak does not exist, then it is unlikely that simple thresholding will produce a good segmentation. In this case, adaptive thresholding may be a better answer.

BIT-PLANE SLICING

Instead of highlighting gray level images, highlighting the contribution made to total image appearance by specific bits might be desired. Suppose that each pixel in an image is represented by 8 bits. Imagine the image is composed of 8, 1-bit planes ranging from bit plane1-0 (LSB)to bit plane 7 (MSB).

In terms of 8-bits bytes, plane 0 contains all lowest order bits in the bytes comprising the pixels in the image and plane 7 contains all high order bits.

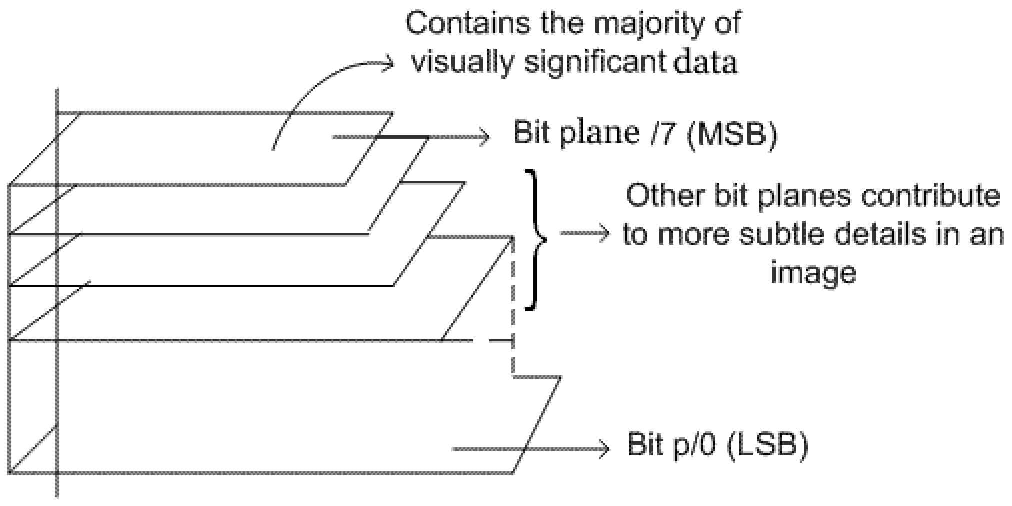

Contains the majority of visually significant data
Bit plane /7 (MSB)
Other bit planes contribute to more subtle details in an image
Bit p/0 (LSB)

Figure 8

Separating a digital image into its bit planes is useful for analyzing the relative importance played by each bit of the image, implying, it determines the adequacy of numbers of bits used to quantize each pixel , useful for image compression.

In terms of bit-plane extraction for a 8-bit image, it is seen that binary image for bit plane 7 is obtained by proceeding the input image with a thresholding gray-level transformation function that maps all levels between 0 and 127 to one level (e.g 0)and maps all levels from 129 to 253 to another (eg. 255).

HISTOGRAM

An "image histogram" is a type of histogram that acts as a graphical representation of the tonal distribution in a digital image.[1] It plots the number of pixels for each tonal value. By looking at the histogram for a specific image a viewer will be able to judge the entire tonal distribution at a glance.

Image histograms are present on many modern digital cameras. Photographers can use them as an aid to show the distribution of tones captured, and whether image detail has been lost to blown-out highlights or blacked-out shadows.[2]

The horizontal axis of the graph represents the tonal variations, while the vertical axis represents the number of pixels in that particular tone.[1] The left side of the horizontal axis represents the black and dark areas, the middle represents medium grey and the right hand side represents light and pure white areas. The vertical axis represents the size of the area that is captured in each one of these zones. Thus, the histogram for a very dark image will have the majority of its data points on the left side and center of the graph. Conversely, the histogram for a very bright image with few dark areas and/or shadows will have most of its data points on the right side and center of the graph.

Image Enhancement by histogram modification

The histogram of an image represents the relative frequency of occurrence of the various gray levels in the image.

It provides a total description of the appearance of an image. The type and degree of enhancement obtained depends on the nature of the specified histogram.

Let the variable *r* represent the grey level of the pixels in the image to be enhanced. Assume that the pixel values are normalized to lie in the range

$0 \leq r \leq 1$ with $r = 0$ represents black

$T(r)$ represents white in the gray scale

For any *r* in [*0,1*], we consider transformations of the form $S = T(r)$ which produce a level *S* for every pixel value *r*in the original image. It is assumed that the transformation function satisfies the conditions:

(1) $T(r)$) is singled valued and monotonically increasing in the interval { $0 \leq r \leq 1$};

(2) $0 \leq T(r) \leq 1$ *, for* $0 \leq r \leq 1$

Condition (1) transformation preserves the order from black to white in the gray scale

Condition (2) transformation guarantees a mapping that is consistent with the allowed range of pixel values.

Example of such a transformation is:

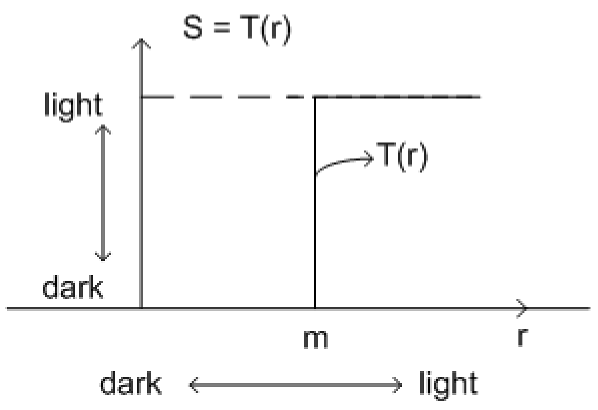


Figure 9

The inverse transform $r = T^{-1}(s)$ for $0 \le s \le 1$, where it is assumed $T^{-1}(s)$ satisfies conditions (1) or (2) wrt variable*s* The gray levels in an image are random quantities in the interval *[0,1]* . Assuming that they are continuous variables the original and transformed gray levels can be characterized by their probability density functions *Pr(r)* and*Ps(s*). A great deal can be said about the general characteristics of an image from the density functions of its gray levels.

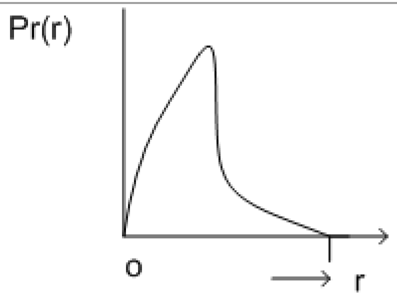
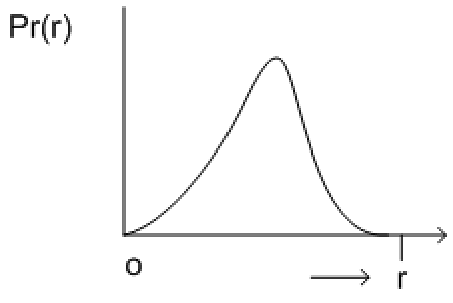
 **This function implies that the image will have dark characters since majority of levels are concentrated inthe dark region of gray scale**	 **The image will have predominant light tones since majority of pixels are light gray.**

Figure 10

It follows from elementary probability theory that if *Pr(r)* and $T(r)$ are known and $T^{-1}(s)$ satisfies condition (1), then

$$P_s(s) = \left[P_r(r) \frac{dr}{ds} \right]_{r = T^{-1}(s)}$$

The enhancement techniques are based on modifying the appearance of an image by controlling the probability density function of its gray levels via the transformation function $T(r)$.

HISTOGRAM EQUALIZATION

The goal is to obtain a uniform histogram for the output image.

This transformation in terms of enhancement implies an increase in the dynamic range of the pixels which can have a considerable effect in the appearance of an image.

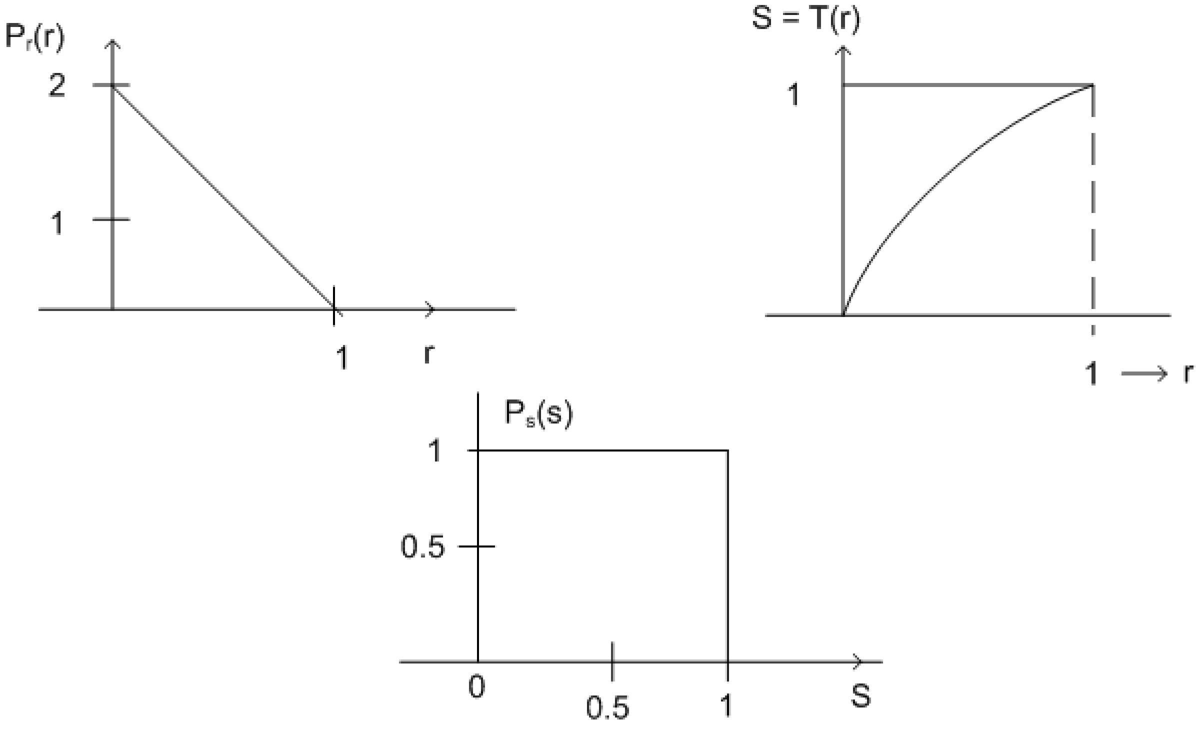


Figure 11

We have,

$$p_r(r) = -2r + 2, \qquad 0 \le r \le 1$$

= 0 elsewhere

$$\therefore\ s = T(r) = \int_0^r (-2w + 2)dw = {}^{-r^2 + 2r}$$

To show: $p_s(s)$ is in fact uniform.

$$r = T^{-1}(s) = 1 \pm \sqrt{1 - s}$$

Since *r* lies in [0,1]

$$r = T^{-1}(s) = 1 - \sqrt{1 - s}$$

$$p_s(s) = \left[p_r(r)\frac{dr}{ds} \right]_{r=T^{-1}(s)=1-\sqrt{1-s}}$$

$$= \left[(-2r+2)\frac{dr}{ds} \right]_{r=1-\sqrt{1-s}}$$

$$= \left[2\sqrt{1-s}\frac{dr}{ds}(1-\sqrt{1-s}) \right] = 1$$,

whish is a uniform density function in the desired image.

CHANGING THE CONTRAST AND BRIGHTNESS OF AN IMAGE.

In this tutorial you will learn how to

- Access pixel values
- Initialize a matrix with zeros
- Learn what saturate_cast does and why it is useful
- Get some cool info about pixel transformations
- In this we enhance the brightness of an image by multiplying each pixel of the image with an alpha value and adding another beta value to it.
- We **OpenCV** function **convertTo** that does the above operation automatically. It can be found under **Mat** package. Its syntax is given below:

```
int alpha = 2;
int beta = 50;
sourceImage.convertTo(destination, rtype , alpha, beta);
```

- The parameters are described below:

Sr.No.	Parameters
1	**destination** It is destination image.

2	**rtype** It is desired output matrix type or, rather the depth, since the number of channels are the same as the input has. if rtype is negative, the output matrix will have the same type as the input.

3	**alpha** It is optional scale factor.
4	**beta** It is optional delta added to the scaled values.

Apart from the convertTo method, there are other methods provided by the Mat class. They are described briefly:

Sr.No.	Methods
1	**adjustROI(int dtop, int dbottom, int dleft, int dright)** It adjusts a submatrix size and position within the parent matrix.
2	**copyTo(Mat m)** It copies the matrix to another one.
3	**diag()** It extracts a diagonal from a matrix, or creates a diagonal matrix.
4	**dot(Mat m)**

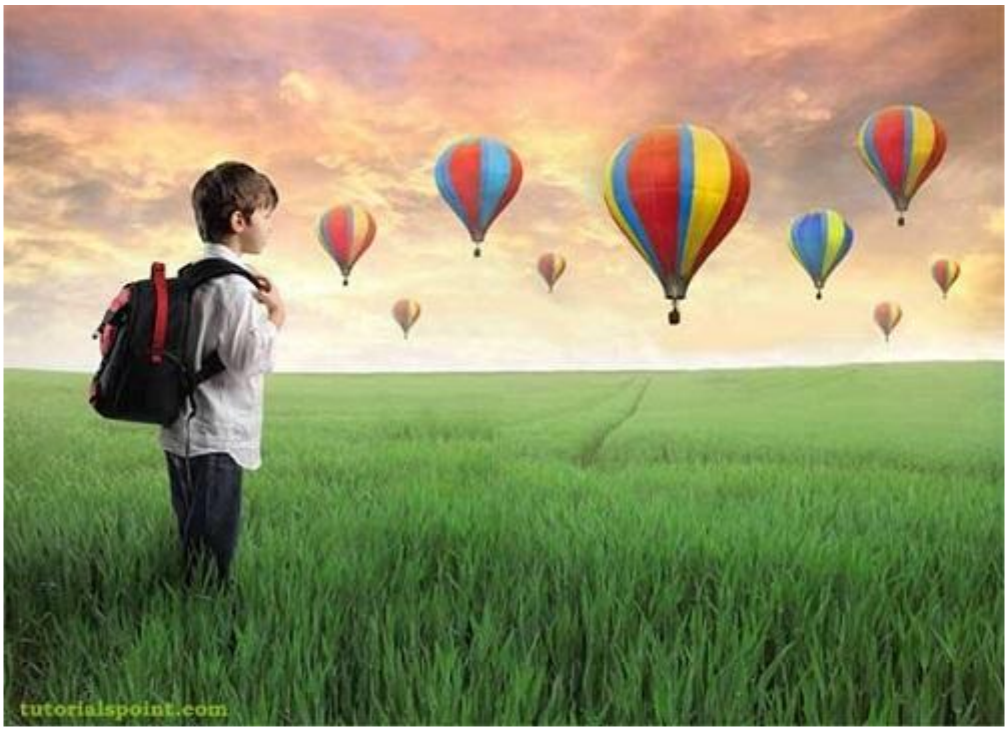
	It computes a dot-product of two vectors.
5	**reshape(int cn)** It changes the shape and/or the number of channels of a 2D matrix without copying the data.
6	**submat(Range rowRange, Range colRange)** It extracts a rectangular sub matrix.

The following example demonstrates the use of Mat class to enhance brightness of an image:

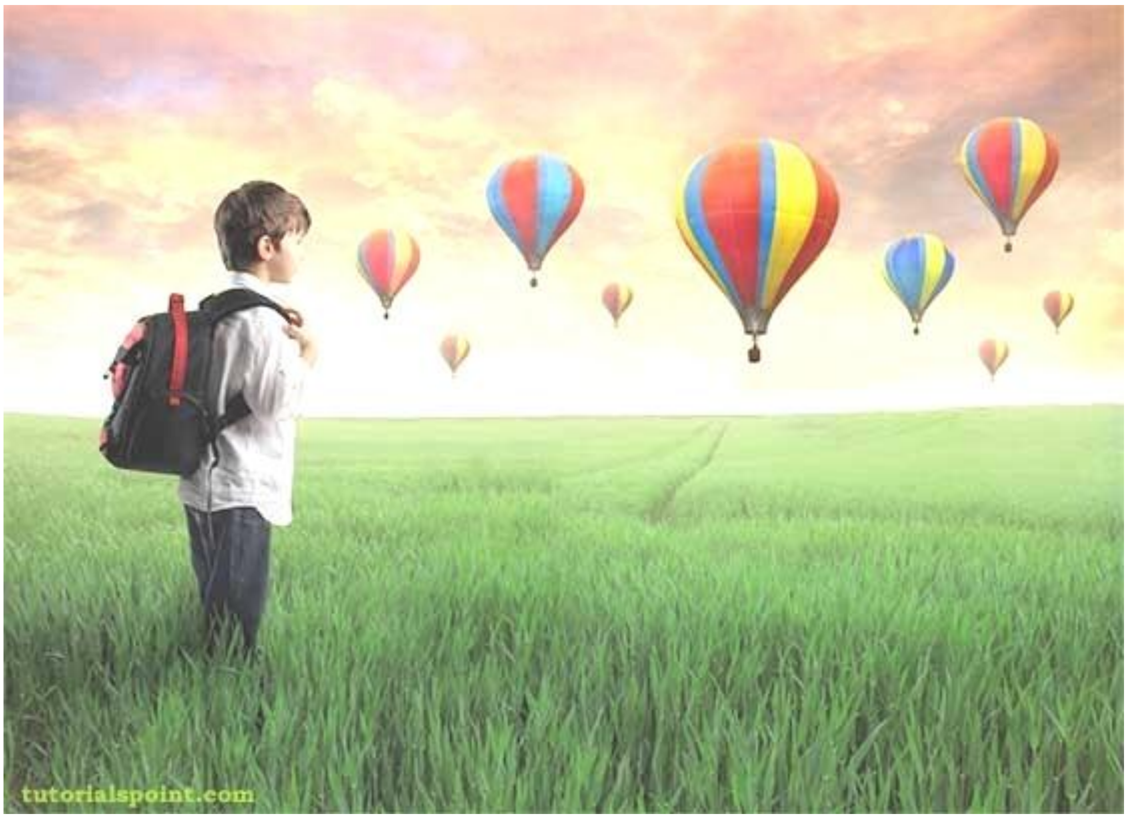
OUTPUT

When you execute the given code, the following output is seen:

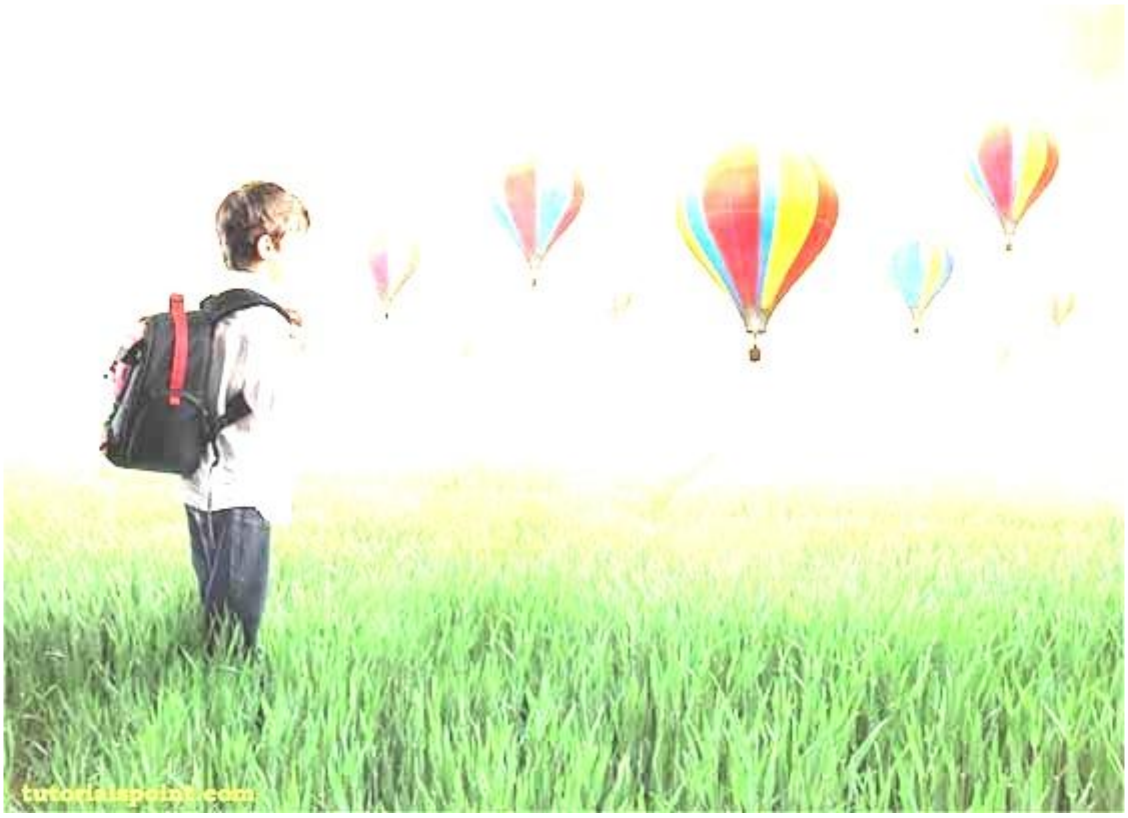
Original Image



Enhanced Bright Image (Alpha=1 & Beta=50)



Enhanced Bright Image (Alpha=2 & Beta=50)



Pixel Transforms

- In this kind of image processing transform, each output pixel’s value depends on only the corresponding input pixel value (plus, potentially, some globally collected information or parameters).
- Examples of such operators include *brightness and contrast adjustments* as well as color correction and transformations.

Brightness and contrast adjustments

- Two commonly used point processes are *multiplication* and *addition* with a constant:

$$g(x) = \alpha f(x) + \beta$$

- The parameters $\alpha > 0$ and β are often called the *gain* and *bias* parameters; sometimes these parameters are said to control *contrast* and *brightness*respectively.

- You can think of $f(x)$ as the source image pixels and $g(x)$ as the output image pixels. Then, more conveniently we can write the expression as:

$$g(i,j) = \alpha \cdot f(i,j) + \beta$$

where i and j indicates that the pixel is located in the *i-th* row and *j-th* column.

1. We begin by creating parameters to save α and β to be entered by the user:

```
2. double alpha;
3. int beta;
```

4. We load an image using imread and save it in a Mat object:

```
5. Mat image = imread( argv[1] );
```

6. Now, since we will make some transformations to this image, we need a new Mat object to store it. Also, we want this to have the following features:

- Initial pixel values equal to zero
- Same size and type as the original image

```
7. Mat new_image = Mat::zeros( image.size(), image.type() );
```

We observe that Mat::zeros returns a Matlab-style zero initializer based on *image.size()* and *image.type()*

8. Now, to perform the operation $g(i,j) = \alpha \cdot f(i,j) + \beta$ we will access to each pixel in image. Since we are operating with RGB images, we will have three values per pixel (R, G and B).

IMAGE COMPRESSION TECHNIQUE

An image can easily be compressed and stored through Java. Compression of image involves converting an image into jpg and storing it.

In order to compress an image, we read the image and convert into BufferedImage object.

Further, we get an ImageWriter from **getImageWritersByFormatName()** method found in the ImageIO class. From this ImageWriter, create an **ImageWriteParam** object. Its syntax is given below:

```
Iterator<ImageWriter> list = ImageIO.getImageWritersByFormatName("jpg");

ImageWriteParam obj = writer_From_List.getDefaultWriteParam();
```

From this ImageWriteParam object, you can set the compression by calling these two methods which are **setCompressionMode()** and **setCompressionQuality()**. Their syntaxes are as given below:

```
obj.setCompressionMode(ImageWriteParam.MODE_EXPLICIT);

obj.setCompressionQuality(0.05f);
```

The setCompressionMode() method takes Mode_EXPLICIT as the parameter. Some of the other MODES are described briefly:

Sr.No.	Modes
1	**MODE_DEFAULT** It is a constant value that may be passed into methods to enable that feature for future writes.
2	**MODE_DISABLED** It is a constant value that may be passed into methods to disable that feature for future writes.
3	**MODE_EXPLICIT** It is a constant value that may be passed into methods to enable that feature for future writes.

Apart from the compressions methods, there are other methods provided by the ImageWriteParam class. They are described briefly:

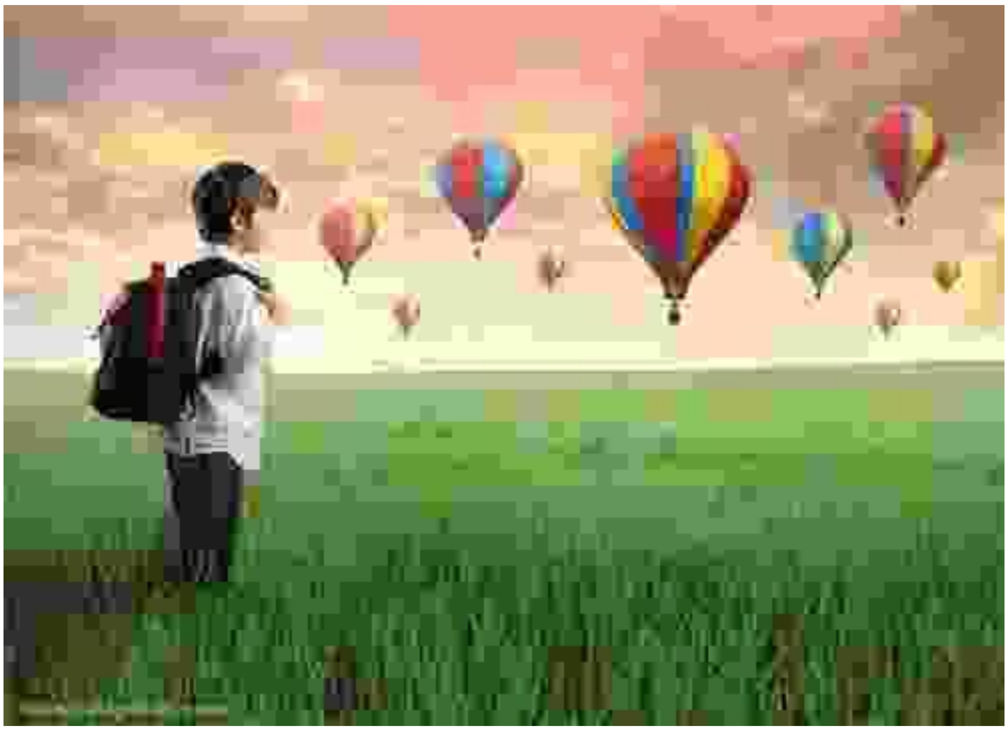
Sr.No.	Methods
1	**canOffsetTiles()** It returns true if the writer can perform tiling with non-zero grid offsets while writing.
2	**getBitRate(float quality)** It returns a float indicating an estimate of the number of bits of output data for each bit of input image data at the given quality level.
3	**getLocale()** It returns the currently set Locale, or null if only a default Locale is supported.
4	**isCompressionLossless()** It returns true if the current compression type provides lossless compression.
5	**unsetCompression()** It removes any previous compression type and quality settings.
6	**unsetTiling()** It removes any previous tile grid parameters specified by calls to setTiling.

Example

The following example demonstrates the use of ImageWriteParam class to compress an image:

<u>Original Image</u>

Compressed Image - Quality Factor: 0.05



Compressed Image - Quality Factor: 0.5

IMAGE SHAPE CONVERSION

The shape of the image can easily be changed by using OpenCV. Image can either be flipped, scaled, or rotated in any of the four directions.

In order to change the shape of the image, we read the image and convert into Mat object. Its syntax is given below:

```
File input = new File("digital_image_processing.jpg");

BufferedImage image = ImageIO.read(input);

//convert Buffered Image to Mat.
```

Flipping an Image

OpenCV allows three types of flip codes which are described below:

Sr.No.	Flip Code
1	**0** 0 means, flipping around x axis.
2	**1** 1 means, flipping around y axis.
3	**-1** -1 means, flipping around both axis.

We pass the appropriate flip code into method **flip()** in the **Core** class. Its syntax is given below:

```
Core.flip(source mat, destination mat1, flip_code);
```

The method **flip()** takes three parameters: the source image matrix, the destination image matrix, and the flip code.

Apart from the flip method, there are other methods provided by the Core class. They are described briefly:

1	**add(Mat src1, Mat src2, Mat dst)** It calculates the per-element sum of two arrays or an array and a scalar.
2	**bitwise_and(Mat src1, Mat src2, Mat dst)** It calculates the per-element bit-wise conjunction of two arrays or an array and a scalar.
3	**bitwise_not(Mat src, Mat dst)** It inverts every bit of an array.

4	**circle(Mat img, Point center, int radius, Scalar color)** It draws a circle.
5	**sumElems(Mat src)** It blurs an image using a Gaussian filter.
6	**subtract(Mat src1, Scalar src2, Mat dst, Mat mask)** It calculates the per-element difference between two arrays or array and a scalar.

<u>ORIGINAL IMAGE</u>

FLIPPED IMAGE

TO CREATE ZOOMING EFFECT

Zooming is the process of enlarging an image so that the details in the image become more visible and prominent.

We use **OpenCV** function **resize** to apply zooming to images. It can be found under**Imgproc** package. Its syntax is given below:

```
Imgproc.resize(source,destination, destination.size(),zoomFactor,zoomFactor,Interpolation);
```

In the resize function, we pass source image, destination image and its size, zooming factor, and the interpolation method to use.

The interpolation methods available are described below:

Sr.No.	**Interpolation methods**

1	**INTER_NEAREST** It is nearest-neighbour interpolation.
2	**INTER_LINEAR** It is bilinear interpolation (used by default).
3	**INTER_AREA** It is resampling using pixel area relation. It may be a preferred method for image decimation, as it gives more-free results.
4	**INTER_CUBIC** It is a bi-cubic interpolation over 4x4 pixel neighbourhood.
5	**INTER_LANCZOS4** It is a Lanczos interpolation over 8x8 pixel neighbourhood.

Apart from the resize method, there are other methods provided by the Imgproc class. They are described briefly:

Sr.No.	Methods
1	**cvtColor(Mat src, Mat dst, int code, int dstCn)** It converts an image from one color space to another.
2	**dilate(Mat src, Mat dst, Mat kernel)** It dilates an image by using a specific structuring element.

3	**equalizeHist(Mat src, Mat dst)** It equalizes the histogram of a grayscale image.
4	**filter2D(Mat src, Mat dst, int ddepth, Mat kernel, Point anchor, double delta)** It convolves an image with the kernel.
5	**GaussianBlur(Mat src, Mat dst, Size ksize, double sigmaX)** It blurs an image using a Gaussian filter.
6	**integral(Mat src, Mat sum)** It calculates the integral of an image.

Original Image

Zoomed Image(Zooming factor: 2)

ERODING AND DILATING

We use **OpenCV** function **erode** and **dilate**. They can be found under **Imgproc** package. Its syntax is given below:

```
Imgproc.erode(source, destination, element);

Imgproc.dilate(source, destination, element);
```

The parameters are described below:

Sr.No.	Parameters
1	**source** It is Source image.
2	**destination** It is destination image.

3	**element** It is a structuring element used for erosion and dilation, if element=Mat(), a 3 x 3 rectangular structuring element is used.

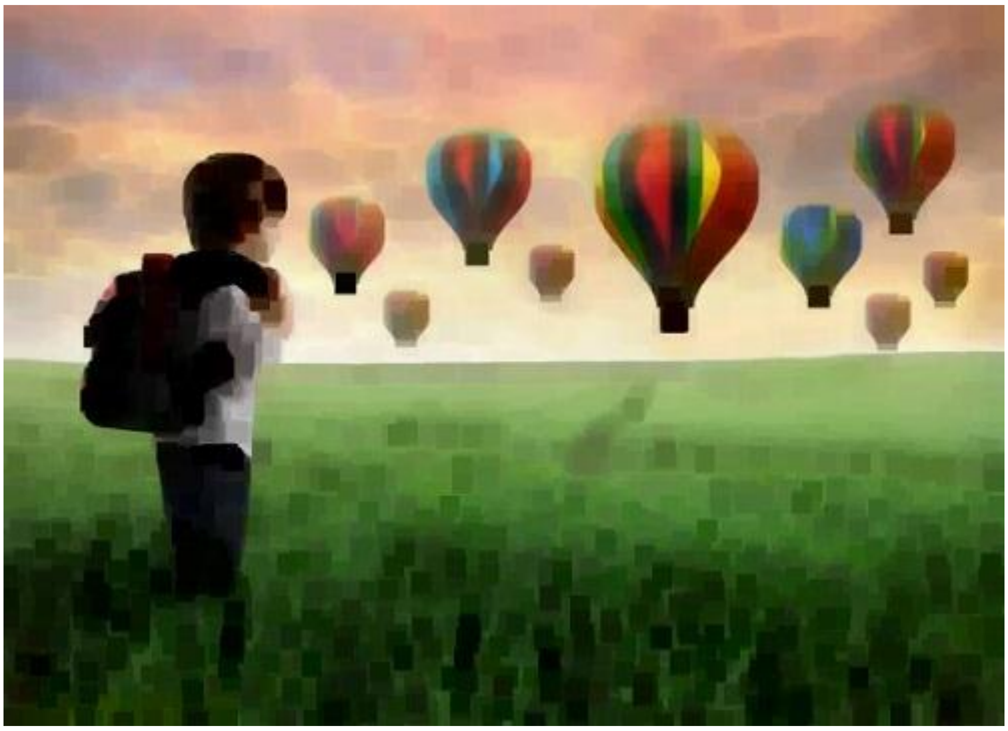
Apart from erode() and dilate() methods, there are other methods provided by the Imgproc class. They are described briefly:

Sr.No.	Method
1	**cvtColor(Mat src, Mat dst, int code, int dstCn)** It converts an image from one color space to another.
2	**dilate(Mat src, Mat dst, Mat kernel)** It dilates an image by using a specific structuring element.
3	**equalizeHist(Mat src, Mat dst)** It equalizes the histogram of a grayscale image.
4	**filter2D(Mat src, Mat dst, int ddepth, Mat kernel, Point anchor, double delta)** It convolves an image with the kernel.
5	**GaussianBlur(Mat src, Mat dst, Size ksize, double sigmaX)** It blurs an image using a Gaussian filter.
6	**integral(Mat src, Mat sum)** It calculates the integral of an image.

Original Image

On the above original image, some erosion and dilation operations have been performed which have been shown in the output below:

Erosion



Dilation

WEIGHTED AVERAGE FILTER

In weighted average filter, we gave more weight to the center value, due to which the contribution of center becomes more than the rest of the values. Due to weighted average filtering, we can control the blurring of image.

We use **OpenCV** function **filter2D** to apply weighted average filter to images. It can be found under **Imgproc** package. Its syntax is given below:

```
filter2D(src, dst, ddepth , kernel, anchor, delta, BORDER_DEFAULT );
```

The function arguments are described below:

Sr.No.	Arguments
1	**src** It is source image.

2	**dst** It is destination image.
3	**ddepth** It is the depth of dst. A negative value (such as -1) indicates that the depth is the same as the source.
4	**kernel** It is the kernel to be scanned through the image.
5	**anchor** It is the position of the anchor relative to its kernel. The location Point(-1, -1) indicates the center by default.
6	**delta** It is a value to be added to each pixel during the convolution. By default it is 0.
7	**BORDER_DEFAULT** We let this value by default.

Apart from the filter2D() method, there are other methods provide by the Imgproc class. They are described briefly:

Sr.No.	Methods
1	**cvtColor(Mat src, Mat dst, int code, int dstCn)** It converts an image from one color space to another.

2	**dilate(Mat src, Mat dst, Mat kernel)** It dilates an image by using a specific structuring element.
3	**equalizeHist(Mat src, Mat dst)** It equalizes the histogram of a grayscale image.
4	**filter2D(Mat src, Mat dst, int ddepth, Mat kernel, Point anchor, double delta)** It convolves an image with the kernel.
5	**GaussianBlur(Mat src, Mat dst, Size ksize, double sigmaX)** It blurs an image using a Gaussian filter.
6	**integral(Mat src, Mat sum)** It calculates the integral of an image.

Original Image

This original image is convolved with the weighted average filter as given below:

Weighted Average Filter

1	1	1
1	10	1
1	1	1

IMAGE PYRAMIDS

Image pyramid is nothing but a method to display a multi-resolution image. The lowermost layer is a highest-resolution version of image and the topmost layer is a lowest-resolution version of the image. Image pyramids are used to handle image at different scales.

In this chapter we perform some down sampling and up sampling on images.

We use **OpenCV** functions **pyrUp** and **pyrDown**. They can be found under **Imgproc**package. Its syntax is given below:

```
Imgproc.pyrUp(source, destination, destinationSize);

Imgproc.pyrDown(source, destination,destinationSize);
```

The parameters are described below:

Sr.No.	**Parameters**

1	**source** It is the source image.
2	**destination** It is the destination image.
3	**destinationSize** It is the size of the output image. By default, it is computed as Size((src.cols*2), (src.rows*2)).

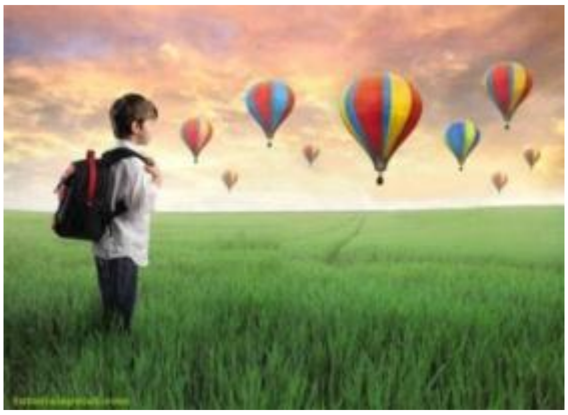
Apart from the pyrUp and pyrDown methods, there are other methods provided by the Imgproc class. They are described briefly:

Sr.No.	**Methods**
1	cvtColor(Mat src, Mat dst, int code, int dstCn) It converts an image from one color space to another.
2	dilate(Mat src, Mat dst, Mat kernel) It dilates an image by using a specific structuring
3	equalizeHist(Mat src, Mat dst) It equalizes the histogram of a grayscale image.
4	filter2D(Mat src, Mat dst, int ddepth, Mat kernel, Point anchor, double delta) It convolves an image with the kernel.
5	GaussianBlur(Mat src, Mat dst, Size ksize, double sigmaX)

	It blurs an image using a Gaussian filter.
6	integral(Mat src, Mat sum) It calculates the integral of an image.

<u>Original Image</u>

On the original image, pyrUp(UP Sampling) and pyrDown(Down Sampling) are performed. The output after sampling is as shown below:



PyrDown image

PyrUP Image

CONCLUSION

Image enhancement algorithms offer a wide variety of approaches for modifying images to achieve visually acceptable images. The choice of such techniques is a function of the specific task, image content, observer characteristics, and viewing conditions. The review of Image enhancement techniques in Spatial domain have been successfully accomplished and is one of the most important and difficult component of digital image processing and the results for each method are also discussed. Based on the type of image and type of noise with which it is corrupted, a slight change in individual method or combination of any methods further improves visual quality. In this survey, we focus on survey the existing techniques of image enhancement, which can be classified into two broad categories as spatial domain enhancement and Frequency domain based enhancement. We show the existing technique of image enhancement and discuss the advantages and disadvantages of these algorithms. Although we did not discuss the computational cost of enhancement
algorithms it may play a critical role in choosing an algorithm for real-time applications. We also have described recent developments methods of image enhancement and point out promising directions on research for image enhancement in spatial domain for future research. The future scope will be the development of adaptive algorithms for effective image enhancement using Fuzzy Logic and Neural Network.

BIBLIOGRAPHY

- A. K. Jain, Fundamentals of Digital Image Processing. Englewood Cliffs, NJ: Prentice Hall, 1989.
- J.C. Russ, The Image Processing Handbook, CRC Press, Boca Raton, FL., 1992. [9] R Hummel, "Histogram
- Rafael C. Gonzalez, and Richard E. Woods, "Digital Image Processing" , 2002,2nd edition.
- Jinshan Tang Eli Peli, and Scott Acton, "Image Enhancement Using a Contrast Measure in the Compressed Domain", IEEE Signal processing Letters , Vol. 10, NO. 10, October2003
- A. Rizzi, C. Gatta, M. Maggiore, E. Agnelli, D. Ferrari, D. Negri, "Automatic Lightness and Color Adjustment of Visual Interfaces", HCIItaly 2003, Torino (Italy), October 2003.
- M. Abdullah-Al-Wadud, Md. Hasanul Kabir, M. Ali Akber Dewan, and Oksam Chae, "A dynamic histogram equalization for image contrast enhancement", IEEE Trans. Consumer Electron., May 2007,vol. 53, no. 2, pp. 593- 600.
- R. C. Gonzalez and R. E. Woods," Digital Image Processing", Prentice Hall, New Jersey, 2008.
- Wadud, M. Kabir, M. H. Dewan and M. C. Oksam, "A dynamic histogram equalization for image contrast enhancement", IEEE Trans. Consumer Electronic, 2007,vol. 53, no. 2, pp. 593-600.
- R. Maini and H. Aggarwel, "A Comprehensive Review of Image Enhancement Techniques," Journal of Computing, Vol. 2, No. 3, 2010, pp. 8-13.
- K. K. Lavania and R. Shivali, Kumar, "A Comparative Study of Image Enhancement Using Histogram Approach," International Journal of Computer Applications, Vol. 32, No. 5, 2011, pp. 1-6.