CHAPTER 3: IMAGE ENHANCEMENT AND MORPHOLOGICAL OPERATIONS

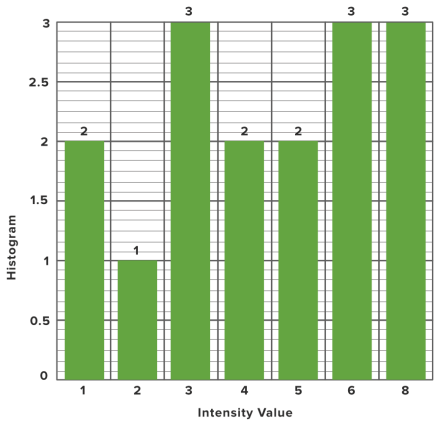
3.1 Contrast and Brightness Adjustments: Histogram equalization

3.1.1 Histogram

The histogram of a digital image with gray levels in the range [0, L-1] is a discrete function.
Histogram Function:
Points about Histogram:
Histogram of an image provides a global description of the appearance of an image.
Information obtained from histogram is very large in quality.
Histogram of an image represents the relative frequency of occurrence of various gray levels in an image.

Let's assume that an Image matrix is given as:

$$\begin{bmatrix} 3 & 6 & 6 & 8 \\ 5 & 3 & 1 & 4 \\ 8 & 6 & 5 & 1 \\ 4 & 8 & 2 & 3 \end{bmatrix}$$



This image matrix contains the pixel values at (i, j) position in the given x-y plane which is the 2D image with gray levels.

In the x-axis has gray levels/ Intensity values and the y-axis has the number of pixels in each gray level. The Histogram value representation of the above image is:

Explanation: The above image has 1, 2, 3, 4, 5, 6, and 8 as the intensity values and the occurrence of each intensity value in the image matrix is 2, 1, 3, 2, 2, 3 and 3 respectively so according to intensity value and occurrence of that particular intensity we mapped them into a Graph.

3.1.2 Brightness and Contrast

Brightness

Brightness is a visual perception in which a source appears to be reflecting light. A color screens use three colors i.e., RGB scheme (red, green and blue) the brightness of the screen depends upon the sum of the amplitude of red green and blue pixels, and it is divided by 3.

$$\mu = \frac{(R+G+B)}{3}$$

The perception of brightness depends upon the optical illusions to appear brighter or darker.

When the brightness is decreased, the color appears dull, and when brightness increases, the color is clearer.In mobile devices, when brightness setting is high, device battery drains fast as compare to the low setting.

Contrast

Contrast is a color which makes an object distinguishable. We can say that contrast is determined by the color and brightness of the object.
Contrast is the difference between the maximum and minimum pixel intensity of an image.

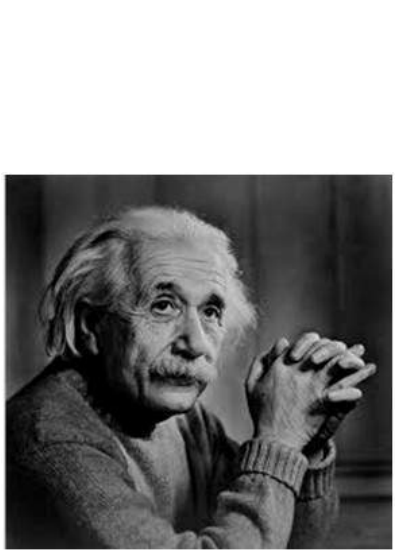
Original image

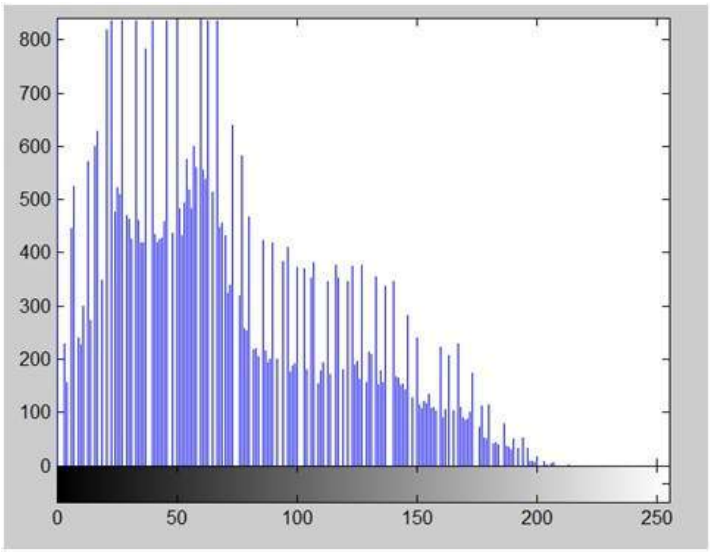
Contrast image

There are two methods of enhancing contrast using histogram

- The first one is called Histogram stretching that increase contrast.
- The second one is called Histogram equalization that enhance contrast

Contrast is the difference between maximum and minimum pixel intensity.
Consider this image.





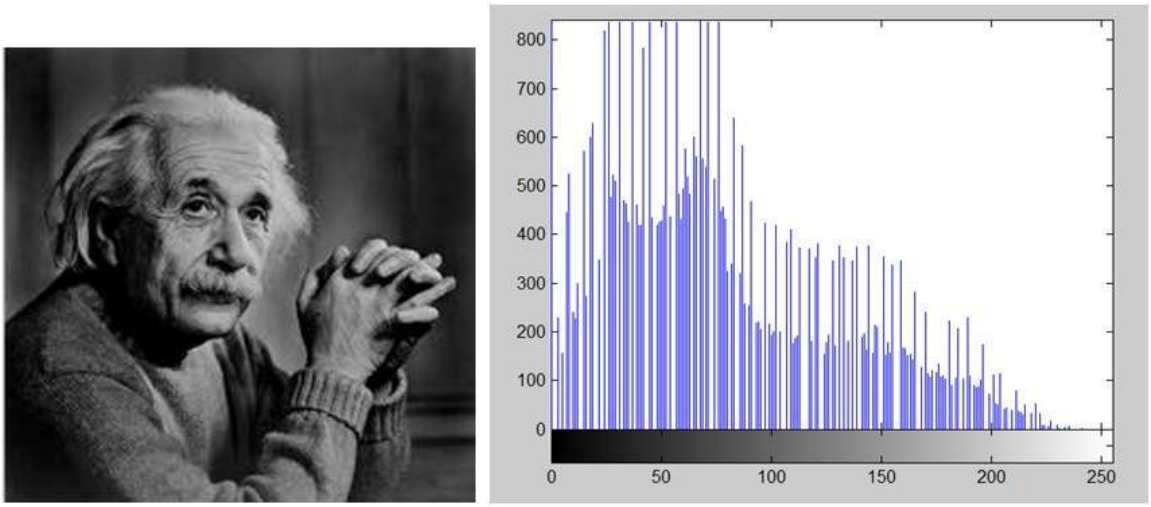
Now we calculate contrast from this image.
Contrast = 225.
Now we will increase the contrast of the image.
Increasing the contrast of the image
The formula for stretching the histogram of the image to increase the contrast is

$$g(x,y) = \frac{f(x,y)-fmin}{fmax- fmin} * 2^{bpp}$$

The formula requires finding the minimum and maximum pixel intensity multiply by levels of gray. In our case the image is 8bpp, so levels of gray are 256.
The minimum value is 0 and the maximum value is 225. So the formula in our case is

$$g(x,y) = \frac{f(x,y)- 0}{225 - 0} * 255$$

where f(x,y) denotes the value of each pixel intensity. For each f(x,y) in an image , we will calculate this formula.
After doing this, we will be able to enhance our contrast.
The following image appear after applying histogram stretching.



The stretched histogram of this image has been shown.
Note the shape and symmetry of histogram.In this case the contrast of the image can be calculated as
Contrast = 240

3.1.3 Histogram Equalization

https://en.wikipedia.org/wiki/Histogram_equalization

Histogram equalization is used to enhance contrast. It is not necessary that contrast will always be increase in this. There may be some cases were histogram equalization can be worse. In that cases the contrast is decreased.
Algorithm:

- Find the frequency of each value represented on the horizontal axis of the histogram i.e. intensity in the case of an image.
- Calculate the probability density function for each intensity value.
- After finding the PDF, calculate the cumulative density function for each intensity's frequency.
- The CDF value is in the range 0-1, so we multiply all CDF values by the largest value of intensity i.e. 255.
- Round off the final values to integer values.

A 3-bit image of size 4×5 is shown below. Compute the histogram equalized image.

0	1	1	3	4
7	2	5	5	7
6	3	2	1	1
1	4	4	2	1

Steps:

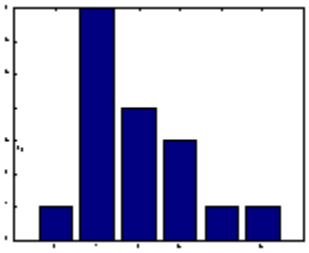
1. Range of intensity values = [0, 1, 2, 3, 4, 5, 6, 7]
2. Frequency of values = [1, 6, 3, 2, 3, 2, 1, 2]
3. total = 20 = 4*5
4. Calculate PDF = frequency of each intensity/Total sum of all frequencies,
5. Calculate CDF =cumulative frequency of each intensity
6. value = sum of all PDF value (<=i)
7. Multiply CDF with 7.
8. Round off the final value of intensity.

Range	Frequency	PDF	CDF	7*CDF	Round-off
0	1	0.0500	0.0500	0.3500	0
1	6	0.3000	0.3500	2.4500	2
2	3	0.1500	0.5000	3.5000	4
3	2	0.1000	0.6000	4.2000	4
4	3	0.1500	0.7500	5.2500	5
5	2	0.1000	0.8500	5.9500	6
6	1	0.0500	0.9000	6.3000	6
7	2	0.1000	1.0000	7.0000	7

Que:
An image histogram is a graphic representation of the frequency counts of all allowable pixel intensities. Figure shows a sample of an image with a skewed histogram.

4	1	3	2
3	1	1	1
0	1	5	2
1	1	2	2

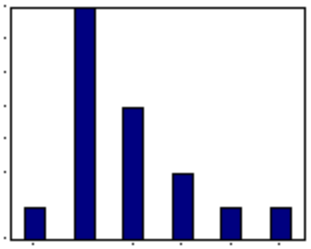
input image



Ans:

5	3	4	4
4	3	3	3
1	3	5	4
3	3	4	4

output image



```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Load the grayscale image
image = cv2.imread('your_image.jpg', cv2.IMREAD_GRAYSCALE)
# Apply Histogram Equalization using OpenCV
equalized_image = cv2.equalizeHist(image)
# Display the original and equalized images
cv2.imshow('Original Image', image)
cv2.imshow('Equalized Image', equalized_image)
# Wait for a key press and close the windows
cv2.waitKey(0)
cv2.destroyAllWindows()
# Plot the histograms
plt.figure(figsize=(10,5))
plt.subplot(1, 2, 1)
plt.hist(image.ravel(), 256, [0,256])
plt.title('Original Histogram')
plt.subplot(1, 2, 2)
plt.hist(equalized_image.ravel(), 256, [0,256])
plt.title('Equalized Histogram')
plt.show()
```

3.2 Image Smoothing

https://www.naukri.com/code360/library/smoothing-images
https://www.scaler.com/topics/smoothing-in-image-processing/
Till now we have discussed two important methods to manipulate images.

1.Transformation functions

This method is known as transformations, in which we discussed different type of transformations and some gray level transformations

2.Graphs (Histograms)

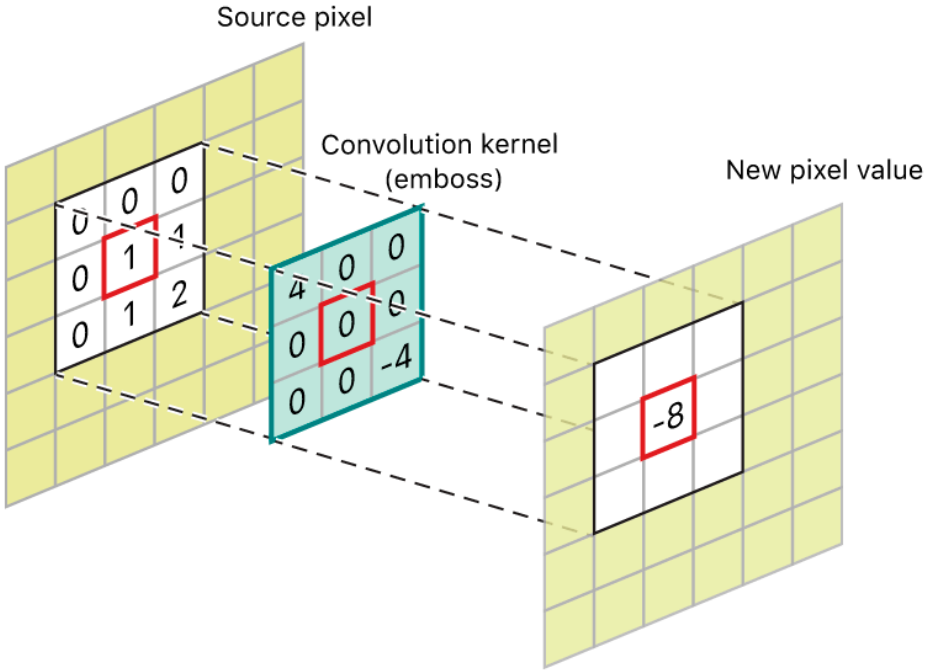
This method is known as histogram processing. We have discussed it in detail for increase contrast, image enhancement, brightness e.t.c

3.2.1 Convolution

Another way of dealing images: 3. Convolution

This other method is known as convolution. So now we are going to use this third method. It can be represented as.

In simple terms, convolution is simply the process of taking a small matrix called the kernel and running it over all the pixels in an image. At every pixel, we'll perform some math operation involving the values in the convolution matrix and the values of a pixel and its surroundings to determine the value for a pixel in the output image.



It can be mathematically represented as
g(x,y) = h(x,y) * f(x,y)
It can be explained as the "mask convolved with an image".
Or
g(x,y) = f(x,y) * h(x,y)
It can be explained as "image convolved with mask".

There are two ways to represent this because the convolution operator(*) is commutative. The h(x,y) is the mask or filter.

What is mask?

Mask is also a signal. It can be represented by a two dimensional matrix. The mask is usually of the order of 1x1, 3x3, 5x5, 7x7 . A mask should always be in odd number, because other wise you cannot find the mid of the mask. Why do we need to find the mid of the mask. The answer is we replace center pixel everytime,check below…

How to perform convolution?

In order to perform convolution on an image, following steps should be taken.

1. Flip the mask (horizontally and vertically) only once
2. Slide the mask onto the image.
3. Multiply the corresponding elements and then add them
4. Repeat this procedure until all values of the image has been calculated.

Example of convolution
Let's perform some convolution. Step 1 is to flip the mask.
Mask: Let's take our mask to be this.

1	2	3
4	5	6
7	8	9

Flipping the mask horizontally

3	2	1
6	5	4
9	8	7

Flipping the mask vertically

9	8	7
6	5	4
3	2	1

Image: Let's consider an image to be like this

2	4	6
8	10	12
14	16	18

(Now this is Convolution)
Step 2: Convolving mask over image. It is done in this way. Place the center of the mask at each element of an image. Multiply the corresponding elements and then add them , and paste the result onto the element of the image on which you place the center of mask.

The box in red color is the mask, and the values in the orange are the values of the mask. The black color box and values belong to the image. Now for the first pixel of the image, the value will be calculated as

First pixel = (5*2) + (4*4) + (2*8) + (1*10)

= 10 + 16 + 16 + 10

= 52

Place 52 in the original image at the first index and repeat this procedure for each pixel of the image.

Why Convolution? Convolution can achieve blurring, sharpening, edge detection, noise reduction e.t.c.

Image smoothing, Noise cleaning, edge enhancement and boundary detection are local operations in image processing. A Local operation is an operation that transfer input image into an output image depending on the neighborhood values of input coordinates. Which means, if pixel b[x,y] is a pixel in output image, it will be result based on the a[x,y], the same pixel in input image and the neighborhood pixel values.

3.2.2 Image blurring

Mean Filter

Mean Filtering is also called as Neighborhood averaging because, in the mean filtering output of a pixel is calculated by getting the average value of neighborhood pixels based on the kernel size.

23	25	30	35	30
25	30	35	37	40
45	40	37	43	45
38	40	43	42	46
35	40	42	45	47

		39		

Value at (3,3) = (30+35+37+40+37+43+40+43+42)/9 = 39

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

or 1/9

1	1	1
1	1	1
1	1	1

1/25

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Python, OpenCV code for mean filtering (kernel size 5x5)

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('lena.jpeg')
kernel = np.ones((5,5),np.float32)/25
dst = cv2.filter2D(img,-1,kernel)
cv2.imshow('avaeraging',dst)
cv2.waitKey(0)
```

Median Filter

In this filter pixel value is replaced by the median value of neighborhood pixels.

23	25	30	35	30
25	30	35	37	40
45	40	37	43	45
38	40	43	42	46
35	40	42	45	47

		40		

Median of 3x3 ,
30, 35, 37, 37, 40, 40, 42, 43, 43

Python, OpenCV code segment for Median Filtering (kernel size 3x3)
median = cv2.medianBlur(img,3)

considering mean and median filtering we can see that median filtering is better.
Median filtering reduce the variance of intensities and preserve the sharpness and location of edges.

Gaussian Filter

Gaussian blur is a mathematical function that blurs images by smoothing out uneven pixel values. It's a common technique in image editing software and graphics applications and is often used to:

- Reduce noise
- Soften text
- Direct the viewer's eye
- Hide features
- Reduce chromatic aberration
- Enhance backgrounds
- Create artistic effects

Gaussian filter is used to blur images and to remove details and noise. As mentioned before mean filter also remove noise and image can be blurred using mean filter. But gaussian filter use a different kernel.
Process to Apply a Gauss filter

1. define the size of the Kernel/Matrix that would be used for demising the image. The sizes are generally odd numbers, i.e. the overall results can be computed on the central pixel. Also the Kernels are symmetric & therefore have the same number of rows and column. The values inside the kernel are computed by the Gaussian function, which is as follows:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Where,
x → X coordinate value
y → Y coordinate value
π → Mathematical Constant PI (value = 3.13)
σ → Standard Deviation

Using the above function a gaussian kernel of any size can be calculated, by providing it with appropriate values. A Gaussian Kernel Approximation(two-dimensional) with Standard Deviation = 1, appears as follows

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

3x3 Gaussian kernel

1/273

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

5x5 Gaussian kernel

2. Convolve filter over image

To calculate the weights of a Gaussian filter kernel with a standard deviation (σ) of 1.5 and a size of 3×3, we follow these steps:

Steps to Calculate the Weights

1. **Gaussian Function**: The Gaussian kernel is defined by:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

   Where:

   - x, y: Pixel offsets from the center.
   - σ: Standard deviation.

2. **Determine Pixel Coordinates**: For a 3×3 kernel, the coordinates are centered around (0, 0):

$$x, y \in \{-1, 0, 1\}$$

3. **Compute Each Weight**: Calculate $G(x, y)$ for each pair of x, y using $\sigma = 1.5$.
4. **Normalize the Kernel**: Sum all values and divide each by the total to ensure the kernel sums to 1.

Step 1: Define the Kernel Centered at (0,0)

For a **3x3 kernel**, the coordinates (x, y) range from -1 to 1, where:

- $x, y = \{-1, 0, 1\}$.

The kernel grid is:

$$\begin{bmatrix} (-1,-1) & (-1,0) & (-1,1) \\ (0,-1) & (0,0) & (0,1) \\ (1,-1) & (1,0) & (1,1) \end{bmatrix}$$

Step 2: Apply the Gaussian Formula

Substitute the values of x, y and $\sigma = 1.5$ into the Gaussian formula:

$$G(x,y) = \frac{1}{2\pi(1.5)^2} e^{-\frac{x^2+y^2}{2(1.5)^2}}$$

Step 3: Compute Individual Weights

Normalization Factor:

$$\frac{1}{2\pi\sigma^2} = \frac{1}{2\pi(1.5)^2} = \frac{1}{14.1372} \approx 0.0707$$

Exponential Term:

For each point in the kernel grid, compute the exponential factor $e^{-\frac{x^2+y^2}{2(1.5)^2}}$:

- For example, at $x = -1, y = -1$:

$$x^2 + y^2 = (-1)^2 + (-1)^2 = 2$$

$$e^{-\frac{2}{2(1.5)^2}} = e^{-\frac{2}{4.5}} = e^{-0.444} \approx 0.641$$

Multiply by the normalization factor to get the weight.

Here is a table where the Gaussian formula $G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$ is applied to each grid point of the **3x3 kernel** with $\sigma = 1.5$:

(x, y)	$x^2 + y^2$	**Exponential Term** $e^{-\frac{x^2+y^2}{2\sigma^2}}$	**Gaussian Weight** $G(x, y) = \frac{1}{2\pi\sigma^2} \cdot$ Exponential Term
(-1, -1)	$(-1)^2 + (-1)^2 = 2$	$e^{-\frac{2}{4.5}} \approx 0.641$	$\frac{1}{2\pi(1.5)^2} \times 0.641 \approx 0.0453$
(-1, 0)	$(-1)^2 + (0)^2 = 1$	$e^{-\frac{1}{4.5}} \approx 0.801$	$\frac{1}{2\pi(1.5)^2} \times 0.801 \approx 0.0567$
(-1, 1)	$(-1)^2 + (1)^2 = 2$	$e^{-\frac{2}{4.5}} \approx 0.641$	$\frac{1}{2\pi(1.5)^2} \times 0.641 \approx 0.0453$
(0, -1)	$(0)^2 + (-1)^2 = 1$	$e^{-\frac{1}{4.5}} \approx 0.801$	$\frac{1}{2\pi(1.5)^2} \times 0.801 \approx 0.0567$
(0, 0)	$(0)^2 + (0)^2 = 0$	$e^{-\frac{0}{4.5}} = 1.000$	$\frac{1}{2\pi(1.5)^2} \times 1.000 \approx 0.0707$
(0, 1)	$(0)^2 + (1)^2 = 1$	$e^{-\frac{1}{4.5}} \approx 0.801$	$\frac{1}{2\pi(1.5)^2} \times 0.801 \approx 0.0567$
(1, -1)	$(1)^2 + (-1)^2 = 2$	$e^{-\frac{2}{4.5}} \approx 0.641$	$\frac{1}{2\pi(1.5)^2} \times 0.641 \approx 0.0453$
(1, 0)	$(1)^2 + (0)^2 = 1$	$e^{-\frac{1}{4.5}} \approx 0.801$	$\frac{1}{2\pi(1.5)^2} \times 0.801 \approx 0.0567$
(1, 1)	$(1)^2 + (1)^2 = 2$	$e^{-\frac{2}{4.5}} \approx 0.641$	$\frac{1}{2\pi(1.5)^2} \times 0.641 \approx 0.0453$

Step 4: Generate the Full Kernel

Compute weights for all points in the 3x3 grid and normalize the kernel so that the sum of weights equals 1.

I'll perform these calculations step by step. Let's compute.

The calculated **3x3 Gaussian filter kernel** with a standard deviation of 1.5 is as follows (normalized so that the sum of weights equals 1):

$$\begin{bmatrix} 0.0947 & 0.1183 & 0.0947 \\ 0.1183 & 0.1478 & 0.1183 \\ 0.0947 & 0.1183 & 0.0947 \end{bmatrix}$$

```
import cv2

# Load the grayscale image
image = cv2.imread('your_image.jpg', cv2.IMREAD_GRAYSCALE)

# Apply Gaussian Blur with a 5x5 kernel
gaussian_blurred = cv2.GaussianBlur(image, (5, 5), 0)

# Apply Median Filter with a 5x5 kernel
median_filtered = cv2.medianBlur(image, 5)

# Display the original and filtered images
cv2.imshow('Original Image', image)
cv2.imshow('Gaussian Blurred Image', gaussian_blurred)
cv2.imshow('Median Filtered Image', median_filtered)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

3.3 Noise Removal: Bilateral filters, denoising

3.3.1 Noise in images:

Image noise is random variation of brightness or color information in the images captured. It is degradation in image signal caused by external sources.Images containing multiplicative noise have the characteristic that the brighter the area the noisier it. But mostly it is additive. We can model a noisy image as

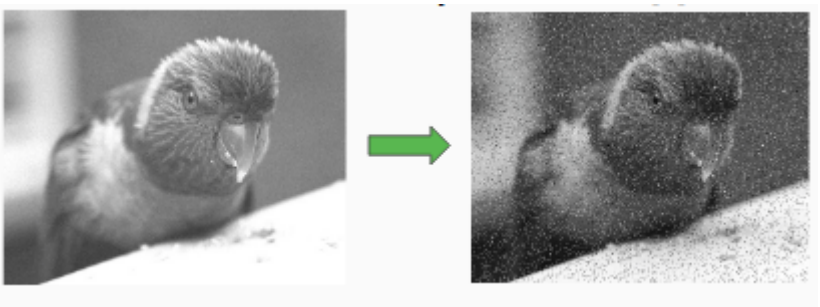


Fig.2 Addition of random noise in an image

A(x,y) = H(x,y) + B(x,y) Where, A(x,y)= function of noisy image, H(x,y)= function of image noise , B(x,y)= function of original image.

Sources of Image noise:

- While image being sent electronically from one place to another.
- Sensor heat while clicking an image.
- With varying ISO Factor which varies with the capacity of camera to absorb light.

Types of Image noise:

There are different types of image noise. They can typically be divided into 3 types.

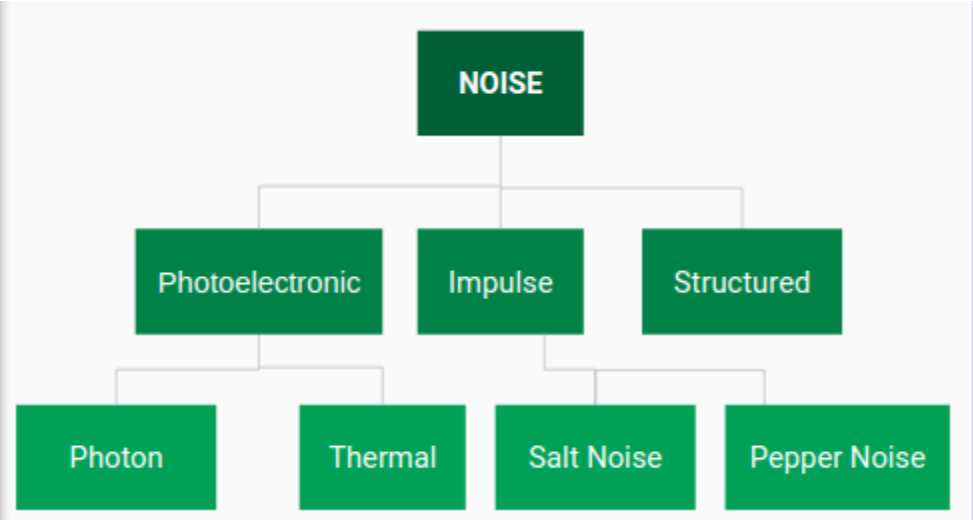
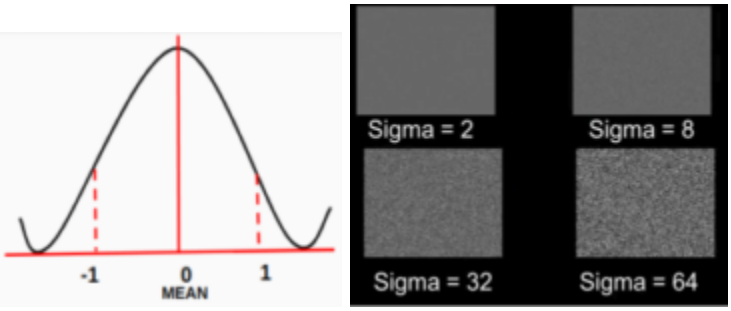


Fig. classification of noise

1. Gaussian Noise:

Gaussian Noise is a statistical noise having a probability density function equal to normal distribution, also known as Gaussian Distribution. Random Gaussian function is added to Image function to generate this noise. It is also called as electronic noise because it arises in amplifiers or detectors.

Source: thermal vibration of atoms and discrete nature of radiation of warm objects.



Plot of Probability Distribution Function

The side image is a bell shaped probability distribution function which have mean 0 and standard deviation(sigma) 1.The magnitude of Gaussian Noise depends on the Standard Deviation(sigma). Noise Magnitude is directly proportional to the sigma value.

The **Gaussian noise equation** models noise as a random variable with a Gaussian (normal) distribution. It is mathematically represented as:

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Explanation of Terms:

1. $P(x)$: The probability density function (PDF) of the noise value x.
2. μ (mean): The average value of the noise. It determines the central point of the distribution.
   - Typically, $\mu = 0$ for zero-mean noise.
3. σ^2 (variance): The spread or variability of the noise.
   - Standard deviation $\sigma = \sqrt{\sigma^2}$.
4. x: The pixel intensity or the noise value.

Implementation of Gaussian Noise with OpenCV-Python:

```
import cv2
import numpy as np

# Function to add Gaussian noise
def add_gaussian_noise(image, mean=0, var=0.01):
    row, col = image.shape
    sigma = var ** 0.5
    gaussian = np.random.normal(mean, sigma, (row, col)) * 255
    noisy_image = image + gaussian.astype(np.uint8)
    noisy_image = np.clip(noisy_image, 0, 255)  # Ensure pixel values remain valid
    return noisy_image.astype(np.uint8)

# Load grayscale image
image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)

# Add Gaussian noise
noisy_image = add_gaussian_noise(image, mean=0, var=0.01)

# Remove noise using Gaussian Blur
denoised_image = cv2.GaussianBlur(noisy_image, (5, 5), 0)

# Display the results
cv2.imshow("Original Image", image)
cv2.imshow("Noisy Image (Gaussian Noise)", noisy_image)
cv2.imshow("Denoised Image (Gaussian Blur)", denoised_image)
```

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```

2. Impulse Noise:

Impulse Function: In the discrete world impulse function on a vale of 1 at a single location and In continuous world impulse function is an idealized function having unit area.

Impulse function in discrete world and continuous world

Types of Impulse Noise:

There are three types of impulse noises. Salt Noise, Pepper Noise, Salt and Pepper Noise.

Salt Noise: Salt noise is added to an image by addition of random bright (with 255 pixel value) all over the image.

Pepper Noise: Salt noise is added to an image by addition of random dark (with 0 pixel value) all over the image.

Salt and Pepper Noise: Salt and Pepper noise is added to an image by addition of both random bright (with 255 pixel value) and random dark (with 0 pixel value) all over the image.This model is also known as data drop noise because statistically it drop the original data values [5]. Source: Malfunctioning of camera's sensor cell.

Implementation of Salt and Pepper Noise with OpenCV-Python:

```
import cv2
import numpy as np

# Function to add salt & pepper noise
def add_salt_pepper_noise(image, prob):
    noisy_image = image.copy()
    row, col = image.shape

    # Number of salt and pepper pixels
    num_salt = int(prob * row * col * 0.5)
    num_pepper = int(prob * row * col * 0.5)

    # Add salt noise (white pixels)
    salt_coords = [np.random.randint(0, i, num_salt) for i in image.shape]
    noisy_image[salt_coords[0], salt_coords[1]] = 255

    # Add pepper noise (black pixels)
```

```
    pepper_coords = [np.random.randint(0, i, num_pepper) for i in image.shape]
    noisy_image[pepper_coords[0], pepper_coords[1]] = 0

    return noisy_image

# Function to remove noise using Median Filter
def remove_noise_with_median_filter(noisy_image, kernel_size=5):
    denoised_image = cv2.medianBlur(noisy_image, kernel_size)
    return denoised_image

# Load a grayscale image
image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)

# Add Salt & Pepper noise with 5% probability
noisy_image = add_salt_pepper_noise(image, prob=0.05)

# Remove noise using Median Filter
denoised_image = remove_noise_with_median_filter(noisy_image, kernel_size=5)

# Display the results
cv2.imshow("Original Image", image)
cv2.imshow("Noisy Image (Salt & Pepper)", noisy_image)
cv2.imshow("Denoised Image (Median Filter)", denoised_image)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

3. Poisson Noise:

The appearance of this noise is seen due to the statistical nature of electromagnetic waves such as x-rays, visible lights and gamma rays. The x-ray and gamma ray sources emitted number of photons per unit time. These rays are injected in patient's body from its source, in medical x rays and gamma rays imaging systems. These sources are having random fluctuation of photons. Result gathered image has spatial and temporal randomness. This noise is also called as quantum (photon) noise or shot noise.

4. Speckle Noise

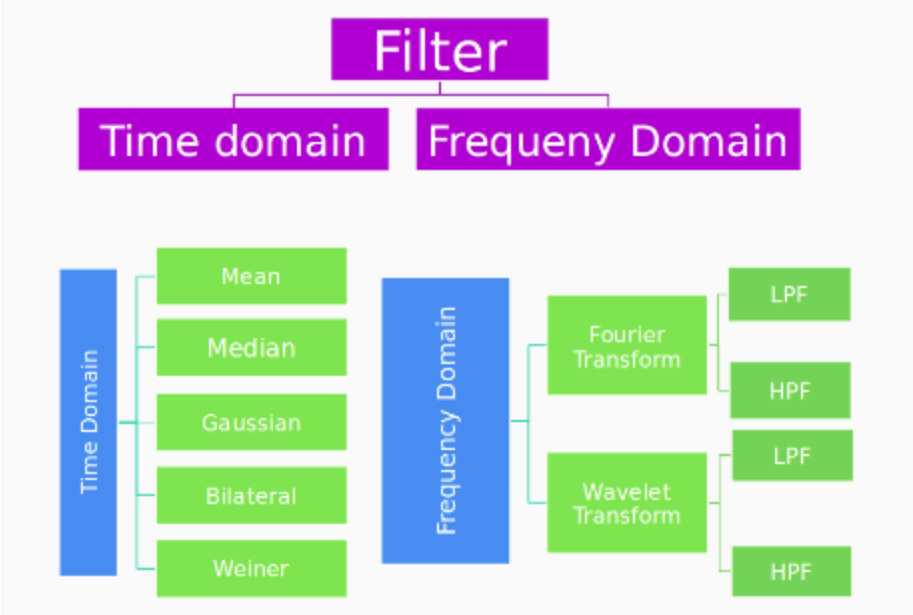
A fundamental problem in optical and digital holography is the presence of speckle noise in the image reconstruction process. Speckle is a granular noise that inherently exists in an image and degrades its quality. Speckle noise can be generated by multiplying random pixel values with different pixels of an image.

```
import cv2
import numpy as np

# Function to add Speckle noise
```

```
def add_speckle_noise(image):
    """
    Add speckle noise to an image.
    Speckle noise is multiplicative noise modeled as I + I * noise.
    """
    row, col = image.shape
    noise = np.random.normal(0, 0.2, (row, col))  # Gaussian noise with mean=0, stddev=0.2
    noisy_image = image + image * noise
    noisy_image = np.clip(noisy_image, 0, 255).astype(np.uint8)
    return noisy_image

# Function to remove speckle noise using Bilateral Filtering
def remove_speckle_noise(image):
    """
    Remove speckle noise using Bilateral Filtering.
    This preserves edges while smoothing the image.
    """
    denoised_image = cv2.bilateralFilter(image, d=9, sigmaColor=75, sigmaSpace=75)
    return denoised_image

# Load a grayscale image
image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)

# Add Speckle noise
noisy_image = add_speckle_noise(image)

# Remove Speckle noise
denoised_image = remove_speckle_noise(noisy_image)

# Display the results
cv2.imshow("Original Image", image)
cv2.imshow("Noisy Image (Speckle)", noisy_image)
cv2.imshow("Denoised Image (Bilateral Filter)", denoised_image)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Types of Image noise filters:

There are different types of image noise filters. They can typically be divided into 2 types.



Classification of Filters

4. Bilateral Filter

Bilateral filtering is a technique used to reduce noise in images while keeping the edges sharp. Unlike other filters (e.g., Gaussian), bilateral filtering considers both the distance between pixels (spatial information) and the difference in pixel values (intensity information).

This helps preserve edges (where there's a significant change in pixel values) while still smoothing areas with little change in pixel values.

How It Works:

1. Spatial Component: The filter looks at nearby pixels (spatial distance). Pixels closer to the target pixel have more influence.
2. Range Component: It also considers how similar the neighboring pixel's intensity is to the target pixel. If the intensity difference is small, the pixel is more likely to influence the target pixel.

Key Idea:

- If two pixels are close together and have similar intensities, they influence each other more.

- If they are far apart or have very different intensities, they influence each other less.

Steps Involved:

1. For each pixel in the image, look at its neighboring pixels.
2. Calculate the weight of each neighboring pixel based on:
    - Spatial distance: How far the pixel is from the center.
    - Intensity difference: How similar the intensity is to the center pixel.
3. Average the neighboring pixel values, weighted by the calculated values.
4. Output the smoothed image where edges are preserved.

If you have a pixel with an intensity of **200** (center pixel) and nearby pixels with values **150**, **250**, and **170**, bilateral filtering will give more weight to the pixel with intensity **170** (as it's closer in value to **200**) and less weight to **150** and **250**.

Bilateral Filtering Equation (Conceptually):

$$I'(x,y) = \frac{\sum_{i,j} I(x+i, y+j) \cdot \text{weight}_{\text{spatial}}(i,j) \cdot \text{weight}_{\text{range}}(I(x+i, y+j) - I(x,y))}{\sum_{i,j} \text{weight}_{\text{spatial}}(i,j) \cdot \text{weight}_{\text{range}}(I(x+i, y+j) - I(x,y))}$$

Where:

- $\text{weight}_{\text{spatial}}$ is based on the distance between pixels.
- $\text{weight}_{\text{range}}$ is based on the difference in pixel intensity.

NOISE	BEST SUITED FILTERS
Salt and Pepper	Median
Poisson	Mean
Gaussian	Gaussian/BilAtERAL
Speckle	Weiner

3.4 Morphological Operations:

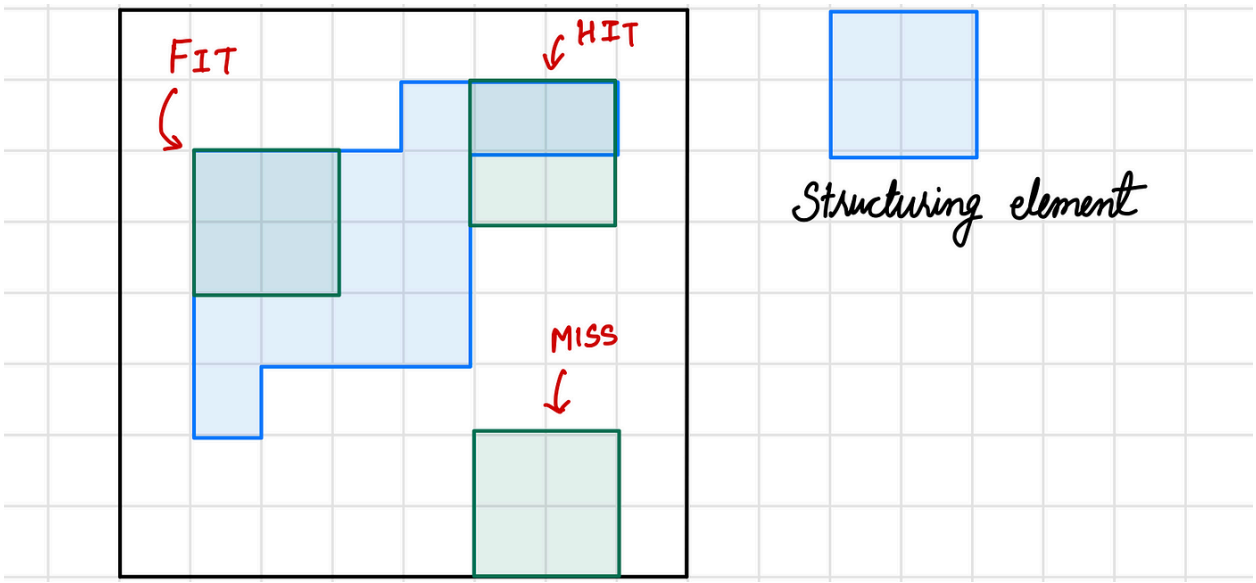
The word 'Morphology' generally represents a branch of biology that deals with the form and structure of animals and plants. However, we use the same term in 'mathematical morphology' to extract image components useful in representing region shape, boundaries, etc.

Morphology is a comprehensive set of image processing operations that process images based on shapes. Morphological operations apply a structuring element to an input image, creating an output image of the same size. In a morphological operation, the value of each pixel in the output image is based on a comparison of the corresponding pixel in the input image with its neighbors. There is a slight overlap between Morphology and Image Segmentation. Morphology consists of methods that can be used to pre-process the input data of Image Segmentation or to post-process the output of the Image Segmentation stage. In other words, once the segmentation is complete, morphological operations can be used to remove imperfections in the segmented image and deliver information on the shape and structure of the image as shown in Figure 2.

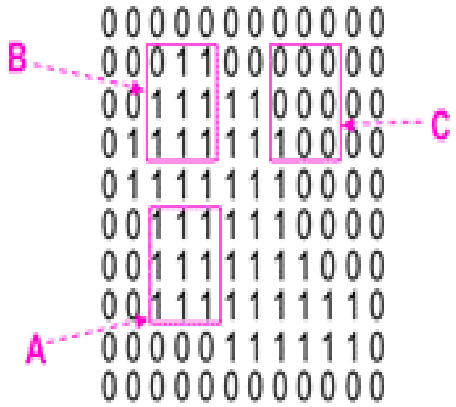
Image after segmentation | Image after segmentation and morphological processing

All morphological processing operations are based on mentioned terms.

- Structuring Element: It is a matrix or a small-sized template that is used to traverse an image. The structuring element is positioned at all possible locations in the image, and it is compared with the connected pixels. It can be of any shape.
- Fit: When all the pixels in the structuring element cover the pixels of the object, we call it Fit.
- Hit: When at least one of the pixels in the structuring element cover the pixels of the object, we call it Hit.
- Miss: When no pixel in the structuring element cover the pixels of the object, we call it miss.



A common practice is to have odd dimensions of the structuring matrix and the origin defined as the centre of the matrix. Stucturing elements play in moprphological image processing the same role as convolution kernels in linear image filtering.
When a structuring element is placed in a binary image, each of its pixels is associated with the corresponding pixel of the neighbourhood under the structuring element. The structuring element is said to fit the image if, for each of its pixels set to 1, the corresponding image pixel is also 1. Similarly, a structuring element is said to hit, or intersect, an image if, at least for one of its pixels set to 1 the corresponding image pixel is also 1.



$$s_1 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \qquad s_2 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

		A	B	C
fit	s_1	yes	no	no
	s_2	yes	yes	no
hit	s_1	yes	yes	yes
	s_2	yes	yes	no

Fitting and hitting of a binary image with structuring elements s1 and s2.

Zero-valued pixels of the structuring element are ignored, i.e. indicate points where the corresponding image value is irrelevant.
The two most widely used operations are Erosion and Dilation.

1. **Erosion**

Check animation:
https://animation.geekosophers.com/morphological-operations/Erosion/Erosion%20Animation%20Js/index.html

Erosion shrinks the image pixels, or erosion removes pixels on object boundaries. First, we traverse the structuring element over the image object to perform an erosion operation, as shown in Figure 4. The output pixel values are calculated using the following equation.

Pixel (output) = 1 {if FIT}

Pixel (output) = 0 {otherwise}

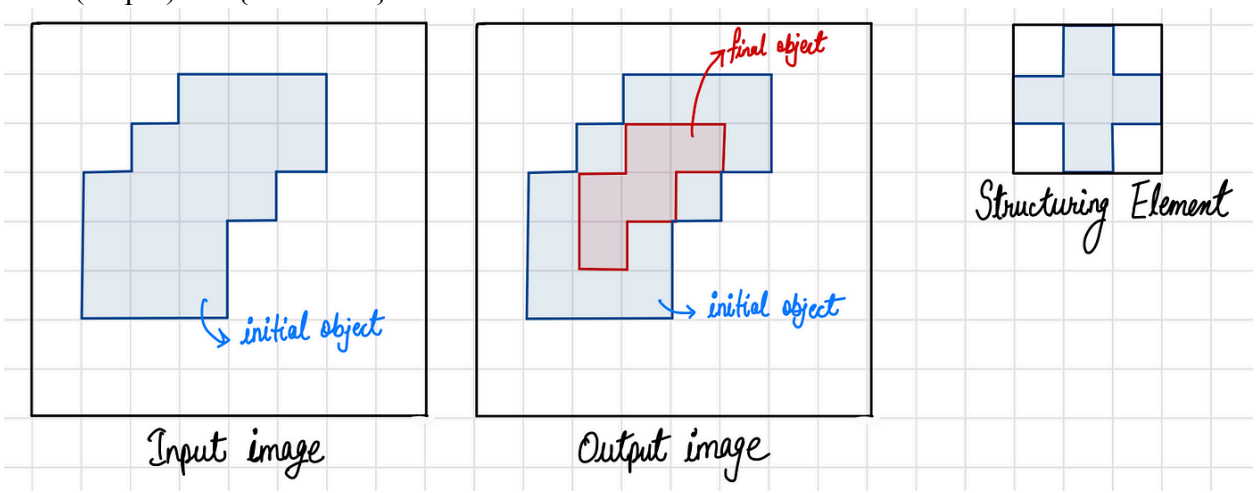


Figure Erosion operation on an input image using a structuring element. .

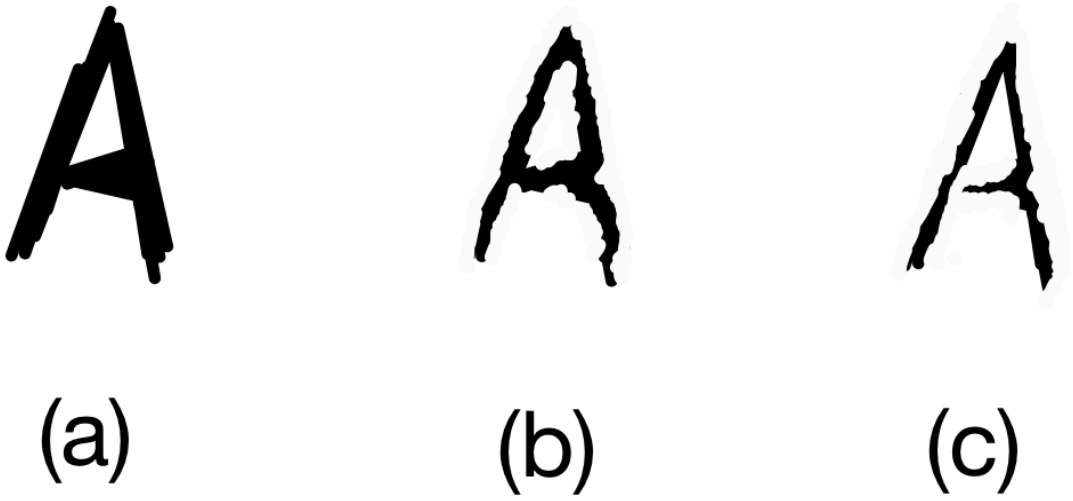


Figure. Results of structuring element size in erosion.

Properties:

It can split apart joint objects (Figure 6).

It can strip away extrusions (Figure 6).

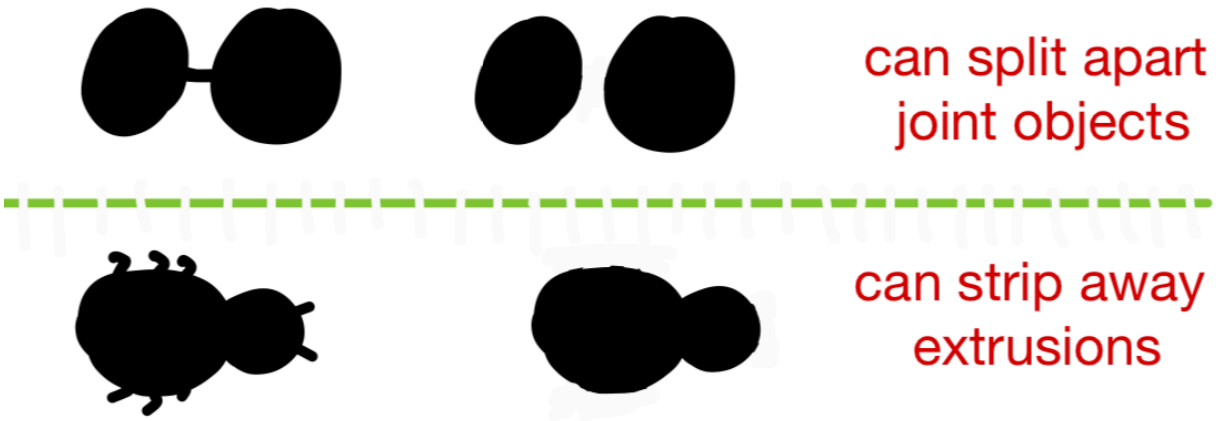
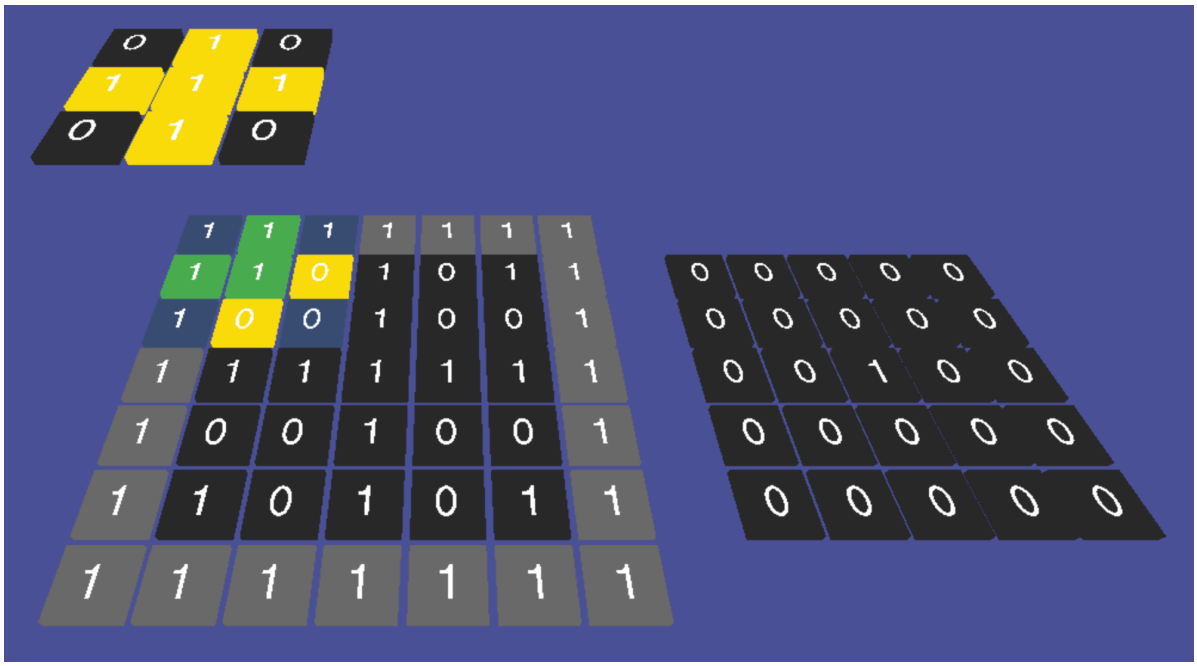


Figure . Example use-cases of Erosion.



2. **Dilation**

https://animation.geekosophers.com/morphological-operations/Dilation/Dilation%20Animation%20Js/index.html

Dilation expands the image pixels, or it adds pixels on object boundaries. First, we traverse the structuring element over the image object to perform an dilation operation, as shown in Figure 7. The output pixel values are calculated using the following equation.
Pixel (output) = 1 {if HIT}
Pixel (output) = 0 {otherwise}

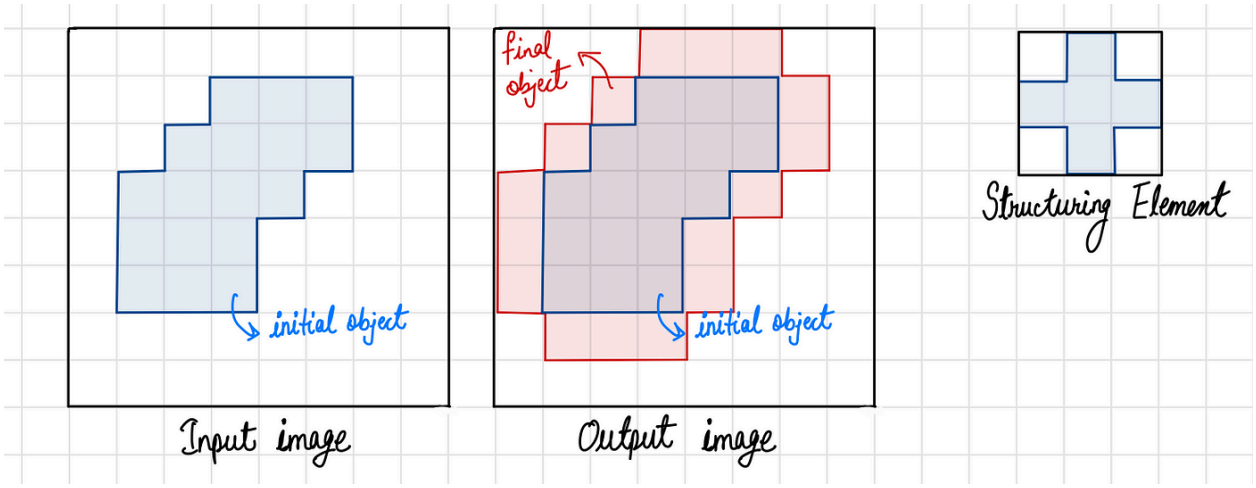


Figure 7. Dilation operation on an input image using a structuring element. (Source: Image by the author)

An example of Dilation is shown in Figure 8. Figure 8(a) represents original image, 8(b) and 8(c) shows processed images after dilation using 3x3 and 5x5 structuring elements respectively.

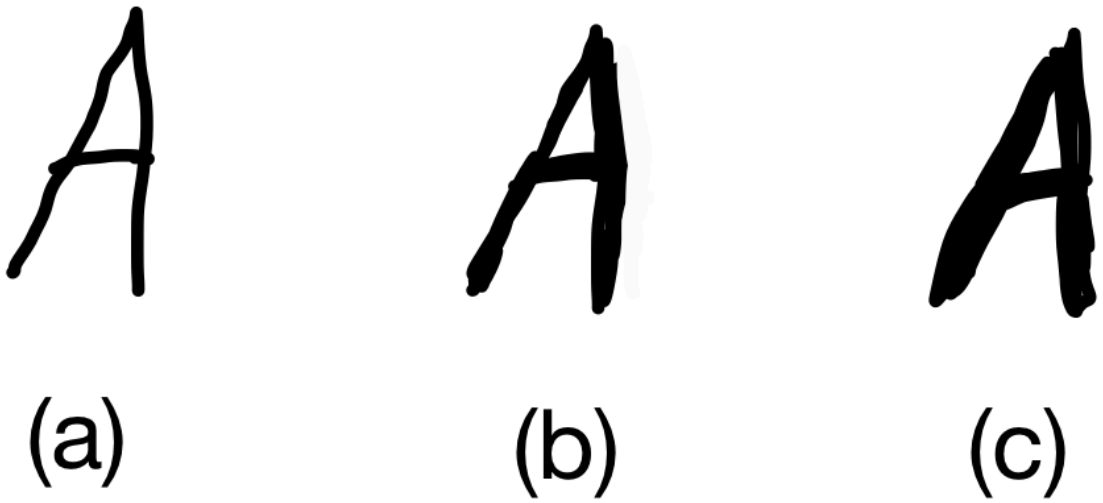


Figure . Results of structuring element size in dilation.

Properties:

It can repair breaks (Figure 9).

It can repair intrusions (Figure 9).

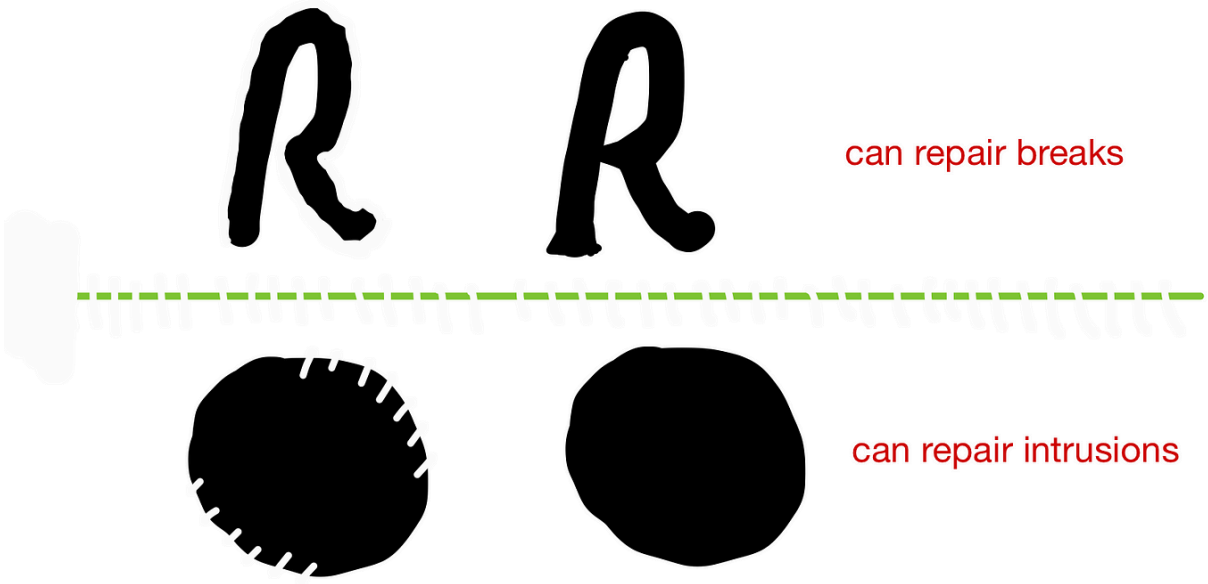
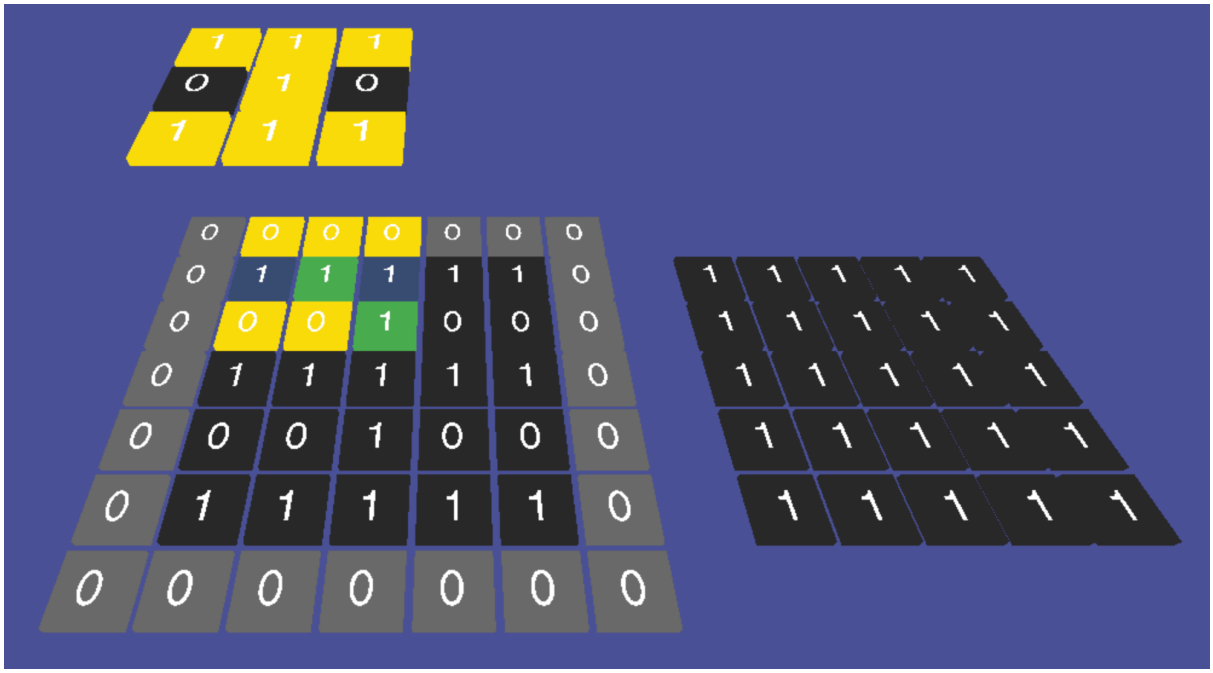


Figure. Example use-cases of DIlation.



3.4.1 Compound Operations

Most morphological operations are not performed using either dilation or erosion; instead, they are performed by using both. Two most widely used compound operations are:

(a) Closing (by first performing dilation and then erosion), and

(b) Opening (by first performing erosion and then dilation). Figure 10 shows both compound operations on a single object.

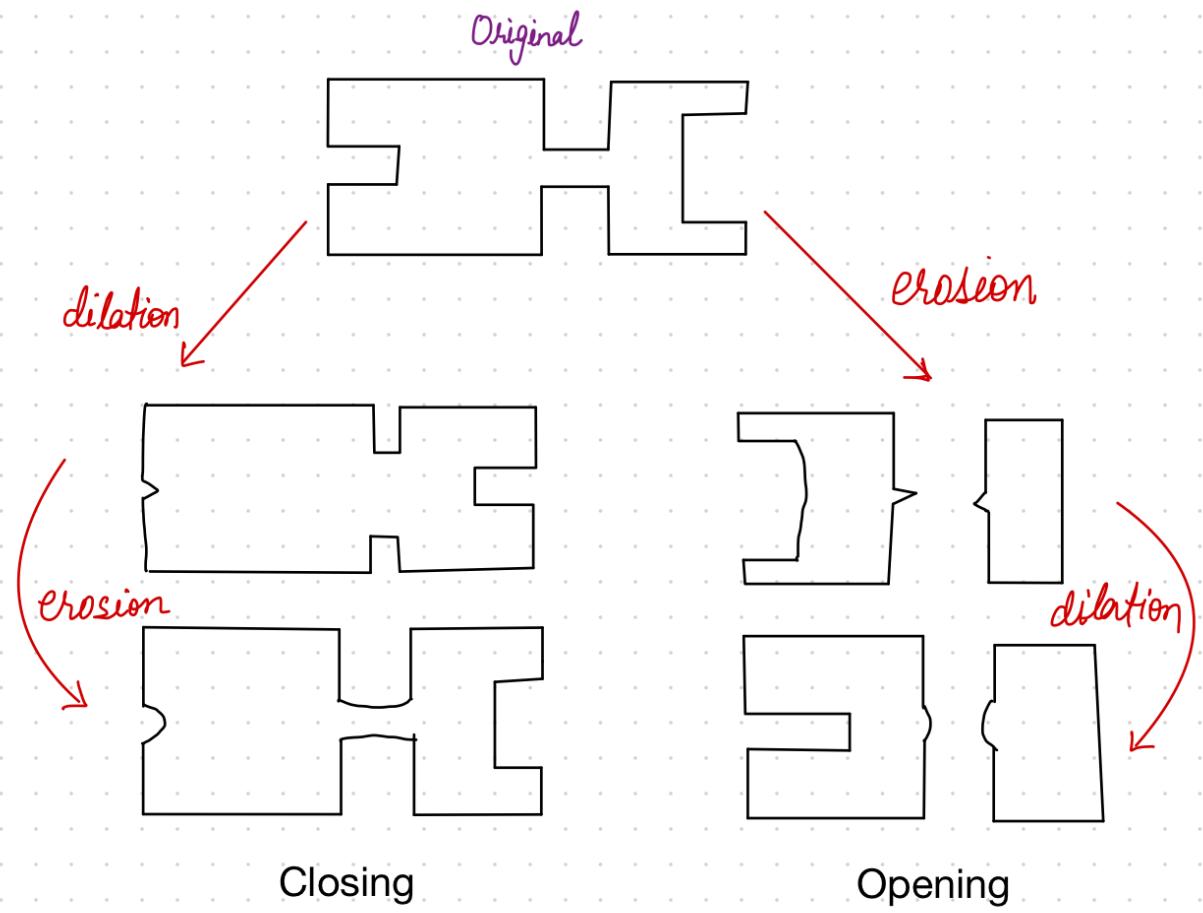


Figure. Output of Compound operations on an input object. (Source: Image by the author)

Opening and **Closing** are two fundamental morphological operations used in image processing. They are based on **erosion** and **dilation**, and their purpose is to modify the structure of objects in an image, often to remove noise, smooth boundaries, or refine shapes.

1. Opening (Erosion followed by Dilation)

Purpose: The **Opening** operation is used primarily to **remove small noise** or **small objects** from the foreground, **smooth the contours** of larger objects, and **break thin connections** between objects. It can be helpful for clearing small irregularities.

Steps for Opening:

- **Step 1: Erosion**
    - **Erosion** shrinks the boundaries of the objects in the image.
    - A structuring element (kernel) is applied to each pixel. If the structuring element fits inside the neighborhood of the pixel, the pixel retains its value; otherwise, it becomes 0.

    - This operation removes small objects or noise, making large objects smaller.
- **Step 2: Dilation**
    - After erosion, **Dilation** is applied.
    - **Dilation** expands the boundaries of the object again.
    - However, since some parts were eroded earlier, only the remaining larger parts will expand, effectively smoothing the shape and filling small gaps that may have been introduced by the erosion.

2. Closing (Dilation followed by Erosion)

Purpose: The **Closing** operation is used to **close small holes or gaps** in the foreground objects and **connect small disjointed regions**. It is commonly used to fill small black holes or remove small noise connected to the background.

Steps for Closing:

- **Step 1: Dilation**
    - **Dilation** expands the boundaries of the objects in the image.
    - The structuring element is applied to each pixel, and if any pixel in the structuring element overlaps the current pixel, the current pixel is set to 1.
    - This operation enlarges the object and fills small gaps or holes inside the object.
- **Step 2: Erosion**
    - After dilation, **Erosion** is applied.
    - **Erosion** shrinks the expanded object back.
    - However, the dilation would have filled small holes, and erosion will not shrink these filled areas back, effectively closing gaps or holes.

---**********---**********---**********---