Dept. of Information and Communication Technology

Islamic University, Bangladesh

Lab Report On

Course No: **ICT-4206**

Course Title: **Digital Image Processing Laboratory**

Submitted To:

Dr. Md. Zahidul Islam

Professor

Dept. of Information & Communication Technology

Submitted By:

Md Bikasuzzaman

Roll No: 1718012

Reg. No: 1323

Session: 2017-18

Faculty of Engineering & Technology

Index

Image: An Image is a spatial representation of a two dimensional or three-dimensional scene. It is an array or a matrix pixel (picture elements) arranged in columns and rows.

Digital image: An image is 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** of that image at that point. When x, y and amplitude values of **F** are finite, we call it a **digital image**.

Digital Image Processing: Digital image processing is the use of computer algorithms to manipulate and analyze digital images. It involves a set of techniques that are used to improve the quality of digital images, extract useful information from them, and transform them into more useful forms.

Grayscale Image: A grayscale image, also known as a black-and-white image, is an image that consists of shades of gray or a single color channel. In a grayscale image, the intensity of each pixel is represented by a single value, typically ranging from 0 (black) to 255 (white), with values in between representing shades of gray. Grayscale images are often used in image processing and computer vision tasks, as they are easier to process and require less memory than full-color images.

RGB image: An RGB image is a digital image that is composed of three color channels: red, green, and blue. These three color channels combine to create a full-color image that can display a wide range of colors and shades. In an RGB image, each pixel is represented by three values, each corresponding to the intensity of one of the three color channels.

OpenCV: OpenCV(Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.

NumPy: NumPy (Numerical Python) is a Python library that provides support for large, multi-dimensional arrays and matrices, along with a large collection of mathematical functions to operate on them. It is one of the core libraries used for scientific computing with Python.

Matplotlib: Matplotlib is a Python library used for creating visualizations, such as charts, graphs, histograms, and scatterplots.

Flags argument of the **cv2.imread()** function, which determines how the image should be read.

1. **cv2.IMREAD_UNCHANGED (-1):** The -1 argument can be used to read the image with its original number of color channels. The resulting array has shape (height, width, channels).
2. **cv2.IMREAD_GRAYSCALE (0):** The image is read as a grayscale image. The resulting array has shape (height, width).
3. **cv2.IMREAD_COLOR (1):** The image is read as a color image, discarding any alpha or transparency channels. The resulting array has shape (height, width, 3).

Have to know for start off the digital image processing:

- Python

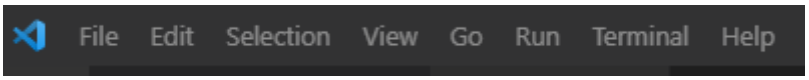
- Data Structures(list, set, tuples, and dictionary)
- Linear Algebra
- Calculus
- Statistics

Environment set up:

Download visual studio code and install it in your PC.

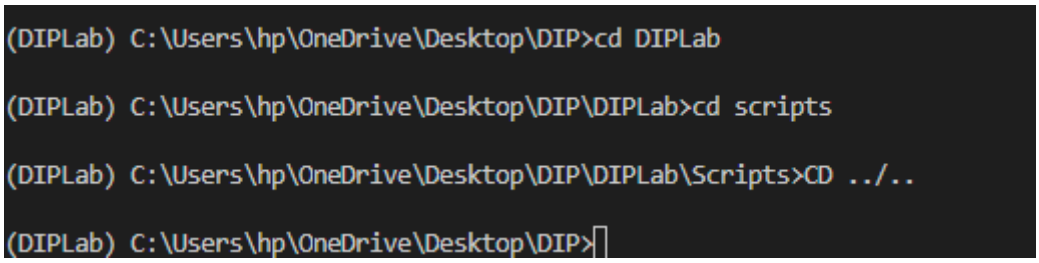
Install opencv in visual studio step by step command:

1. Open Visual studio code.
2. Open terminal.

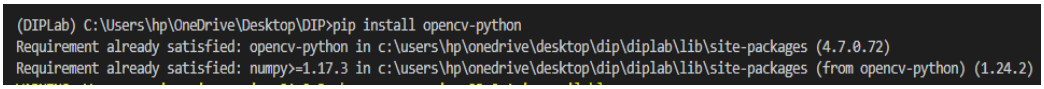


3. Select cmd
4. Create a folder -> **mkdir folder name**
5. Go to the folder -> **cd folder name**
6. Create a virtual environment in the folder -> **python -m venv "virtual environment name"**

7. Go to virtual environment -> **cd "virtual environment name"**
8. Go to Scripts -> **cd Scripts**
9. Write the command -> **activate.bat**
10. Go out from scripts and virtual environment -> **cd ../ ..**



11. Install opencv -> **pip install opencv-python**



Experiment Number: 01

Experiment Name: To create an opencv-python program to get a gradient image from an RGB/grayscale image.

Theory:

Image Gradient: Image is a matrix of pixel values representing various intensity level values. A pixel is the building block of an image. The gradient can be defined as the change in the direction of the intensity level of an image.

RGB Image: An RGB image is a digital image in which each pixel is represented by three color channels: red, green, and blue. Each color channel is represented by an 8-bit value ranging from 0 to 255, indicating the intensity of that color in the pixel. RGB images are commonly used in digital photography, computer graphics, and image processing.

Grayscale image: A grayscale image is a digital image in which each pixel is represented by a single intensity value ranging from 0 to 255. Grayscale images are typically created by taking the average of the red, green, and blue channels of an RGB image.

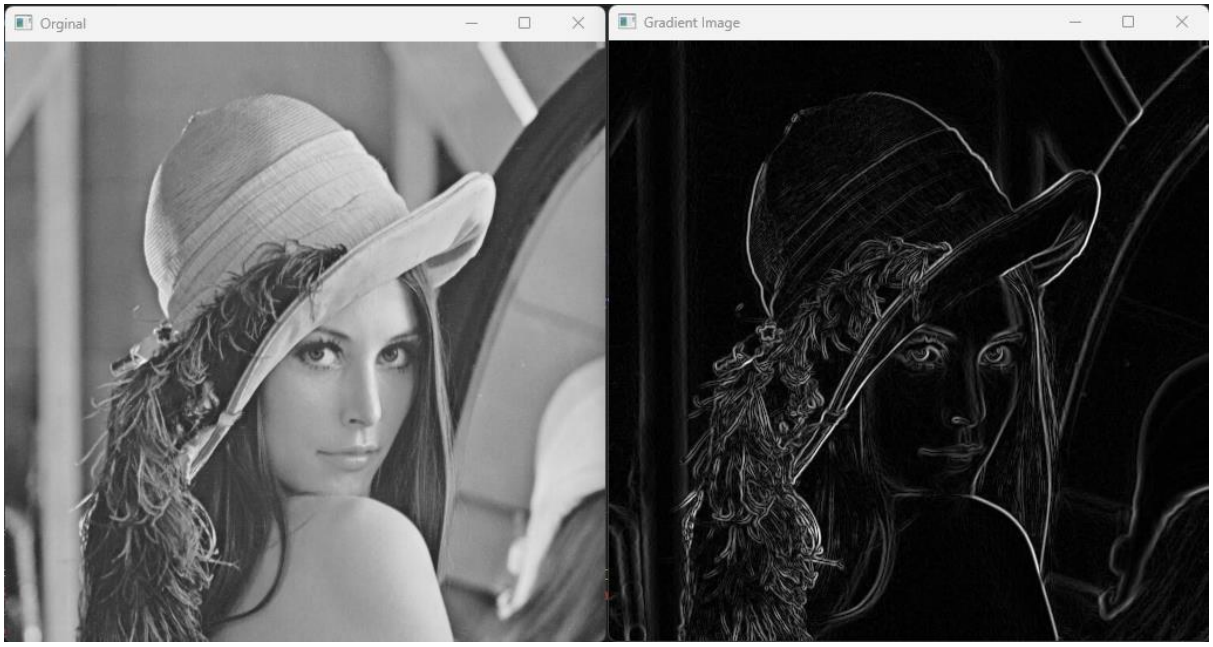
Algorithm:

1. Import the required libraries: cv2 (OpenCV) for computer vision, numpy for numerical operations on images
2. Load the image using **cv2.imread()** method in grayscale format(0), and save it to a variable.
3. Calculate gradient using the sobel operator
4. Calculate the gradient magnitude and direction
5. Display the original and the gradient image using **cv2.imshow()** method.
6. Display the original and gradient images using the **cv2.imshow()** function, and wait for a key press using the **cv2.waitKey()** function before closing the windows with **cv2.destroyAllWindows()**.

Python code:

```
import cv2
# Load image as grayscale
img = cv2.imread("Lenna.png", 0)
# Calculate gradient using the sobel operator
grad_x = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize =3)
grad_Y = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize =3)
# Calculate the gradient magnitude and direction
mag, angle = cv2.cartToPolar(grad_x, grad_Y, angleInDegrees=True)
# Normalize the gradient magnitude to 0-255 range
mag = cv2.normalize(mag, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8U)
# Display the orginal and gradient image
cv2.imshow("Orginal", img)
cv2.imshow("Gradient Image", mag)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Outputs:



Opencv function explanation given below:

1. The **cv2.Sobel()** function to calculate the gradient in the x and y directions using the Sobel operator.
    a. **img:** The input image on which the Sobel filter will be applied. This should be a grayscale image, with each pixel represented by a single intensity value.
    b. **cv2.CV_64F:** It is used to represent 64-bit floating point values.
    c. **1:** The order of the derivative in the x direction. 1 means that the filter will compute the gradient in the horizontal (x) direction.
    d. **0:** The order of the derivative in the y direction. 0 means that the filter will not compute the gradient in the vertical (y) direction.
    e. **ksize=3:** The size of the Sobel kernel. This should be an odd integer value specifying the size of the filter window. ksize=3 specifies a 3x3 filter kernel.
2. The **cv2.cartToPolar()** function to calculate the gradient magnitude and direction. The **angleInDegrees=True** flag is used to get the angle in degrees instead of radians.
3. Normalize the gradient magnitude to the 0-255 range using the **cv2.normalize()** function and the **cv2.NORM_MINMAX** flag.
    a. src: the input array to be normalized
    b. dst: the output array of the same size as src, which stores the normalized values
    c. alpha: the normalization factor applied to the input array
    d. beta: the normalization factor added to the input array
    e. norm_type: the type of normalization to be applied. It can be one of the following values:
        i. cv2.NORM_INF: normalize the input array to the range [0, 1]
        ii. cv2.NORM_L1: normalize the input array so that the absolute sum of its elements is equal to 1

iii. cv2.NORM_L2: normalize the input array so that the sum of the squares of its elements is equal to 1
iv. cv2.NORM_MINMAX: normalize the input array to a user-specified range

f. dtype: the data type of the output array. **cv2.CV_8U**, the **8** indicates that each element of the array or image is represented using 8 bits (i.e., one byte), and the **U** indicates that the elements are unsigned.

Experiment Number: 02

Experiment Name: To create an opencv-python program to show a histogram of an RGB/grayscale image.

Theory:

Histogram: Histogram is a graphical representation of the distribution of pixel intensities in an image. The histogram shows the frequency of occurrence of each intensity value in the image. It is a useful tool for understanding the contrast, brightness, and overall distribution of pixel intensities in an image. It is typically plotted with the intensity values on the x-axis and the number of pixels with that intensity value on the y-axis. The intensity values are usually binned into a fixed number of bins or intervals, with each bin representing a range of intensity values. The height of each bin in the histogram represents the number of pixels in the image with an intensity value falling within that bin's range.

Algorithm:

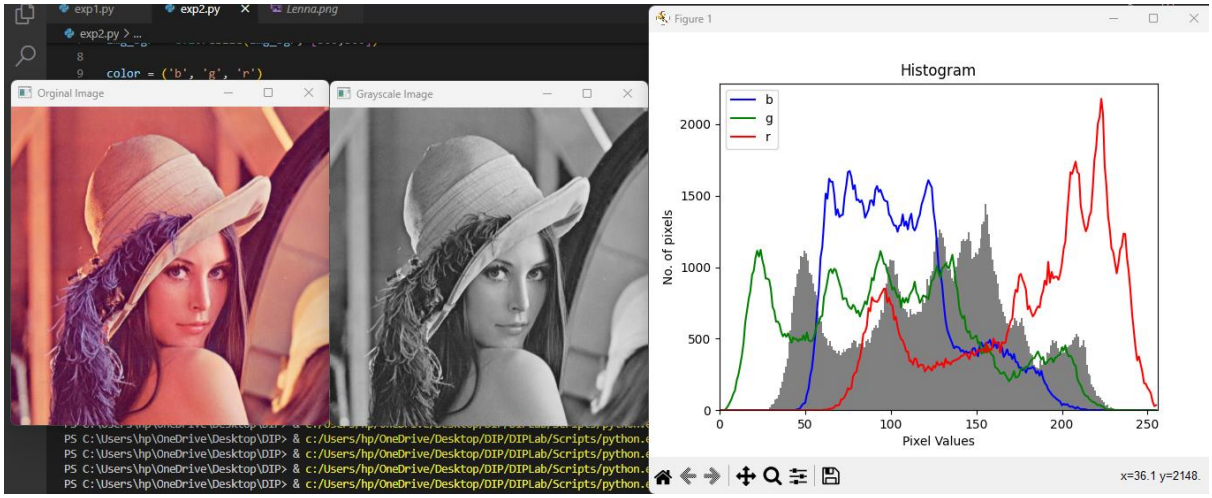
1. Read the image.
2. Compute the histogram
3. Convert image to grayscale using **cv2.cvtColor()**.
4. Pass in the flattened grayscale image **img.ravel()**, specify that we want 256 bins (one for each possible pixel value), and set the range of the histogram to [0, 256].
5. Plot the histogram using **plt.hist().**
6. Call **plt.show()** to display the plot.

Python code:

```
# Experiment no. 02
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Load immage as grayscale
img_bgr = cv2.imread("Lenna.png", 1)
#print(img_bgr.shape)
img_bgr = cv2.resize(img_bgr, [360,360])
color = ('b', 'g', 'r')
for i, col in enumerate(color):
    histr = cv2.calcHist([img_bgr], [i], None, [256], [0, 256])
```

```
    plt.plot(histr, color = col, label = col)
    plt.xlim([0, 256])
gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)
cv2.imshow("Orginal Image", img_bgr)
cv2.imshow("Grayscale Image", gray)
plt.hist(gray.ravel(), 256, [0, 256], color="gray")
plt.title("Histogram")
plt.xlabel("Pixel Values")
plt.ylabel("No. of pixels")
plt.legend()
plt.show()
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Outputs:



Function explanation given below:

- Compute the histogram using **cv2.calcHist()**, which calculates the histogram of a single-channel array. We pass in the grayscale image **img**, specify that we want to calculate the histogram for channel 0 (the only channel in the grayscale image), set the number of bins to 256, and set the range of the histogram to [0, 256] (the range of possible pixel values).
- Plot the histogram using **plt.hist()**, which plots a histogram of a one-dimensional array.

Experiment no: 03

Experiment name: To create an opencv-python program in order to convert a given RGB/grayscale image to a negative image.

Theory:

Negative image: A negative image is an image that has the opposite color values of the original image. A negative color image is additionally color-reversed, with red areas appearing cyan, greens appearing magenta, and blues appearing yellow, and vice versa. In other words, the dark

areas of the original image will be lightened in the negative image, and the light areas will be darkened.

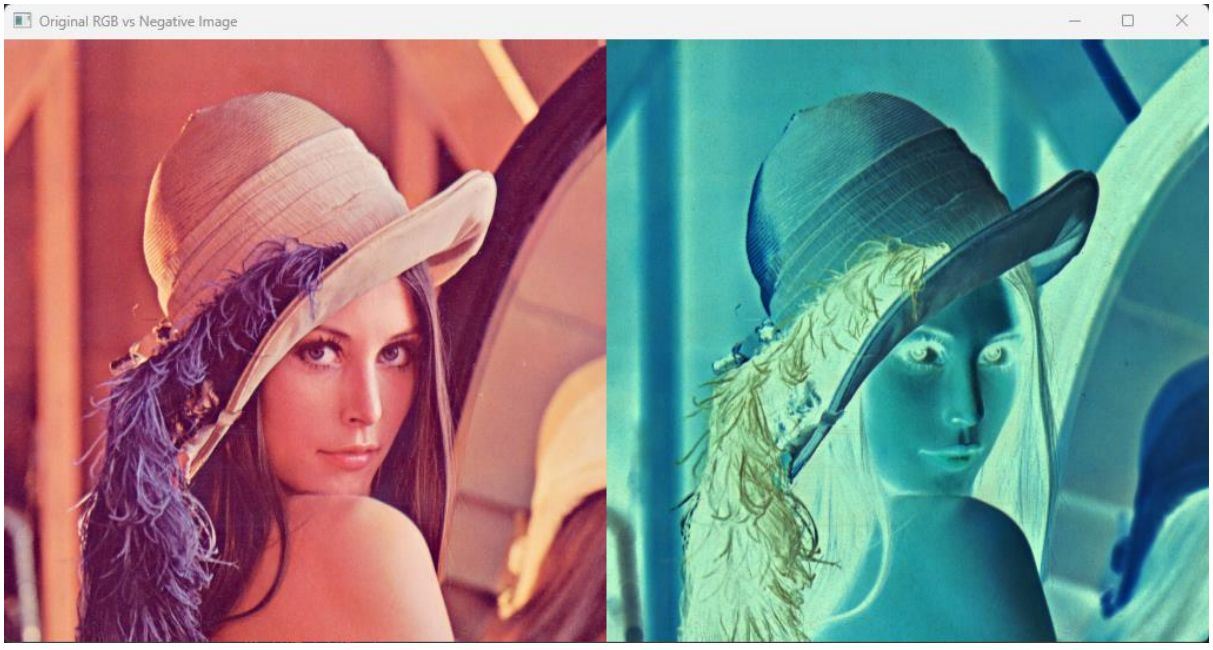
Algorithm:

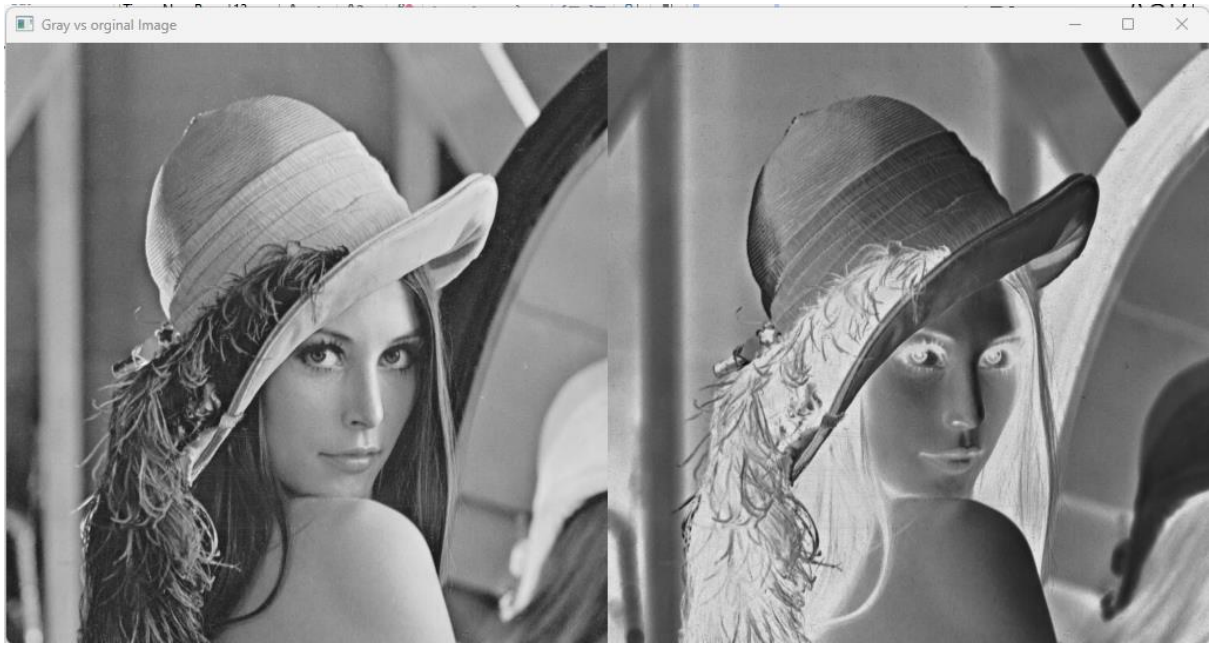
1. Read Image.
2. Convert color from BGR to GRAY.
3. Subtract image array value from 255.
4. Combined original and transformed image.
5. Display Image.

Python code:

```
# Experiment no: 03
import cv2
# Load the image in color mode
img = cv2.imread("Lenna.png")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Invert the color image
neg_img = 255 - img
neg_gray_img = 255 - gray
# Display the original and negative images side by side
combined_img = cv2.hconcat([img, neg_img])
combined_gray_img = cv2.hconcat([gray, neg_gray_img])
cv2.imshow("Original vs Negative Image", combined_img)
cv2.imshow("Gray vs orginal Image", combined_gray_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Outputs:





Experiment no: 04

Experiment name: To create an opencv-python program to enhance color of a given RGB/grayscale image.

Theory:

Enhancement: Improve the visual quality of the image. Process an image so that the result will be more suitable than the original image for a specific application.

Color enhancement: It is a process of improving the color appearance of an image.

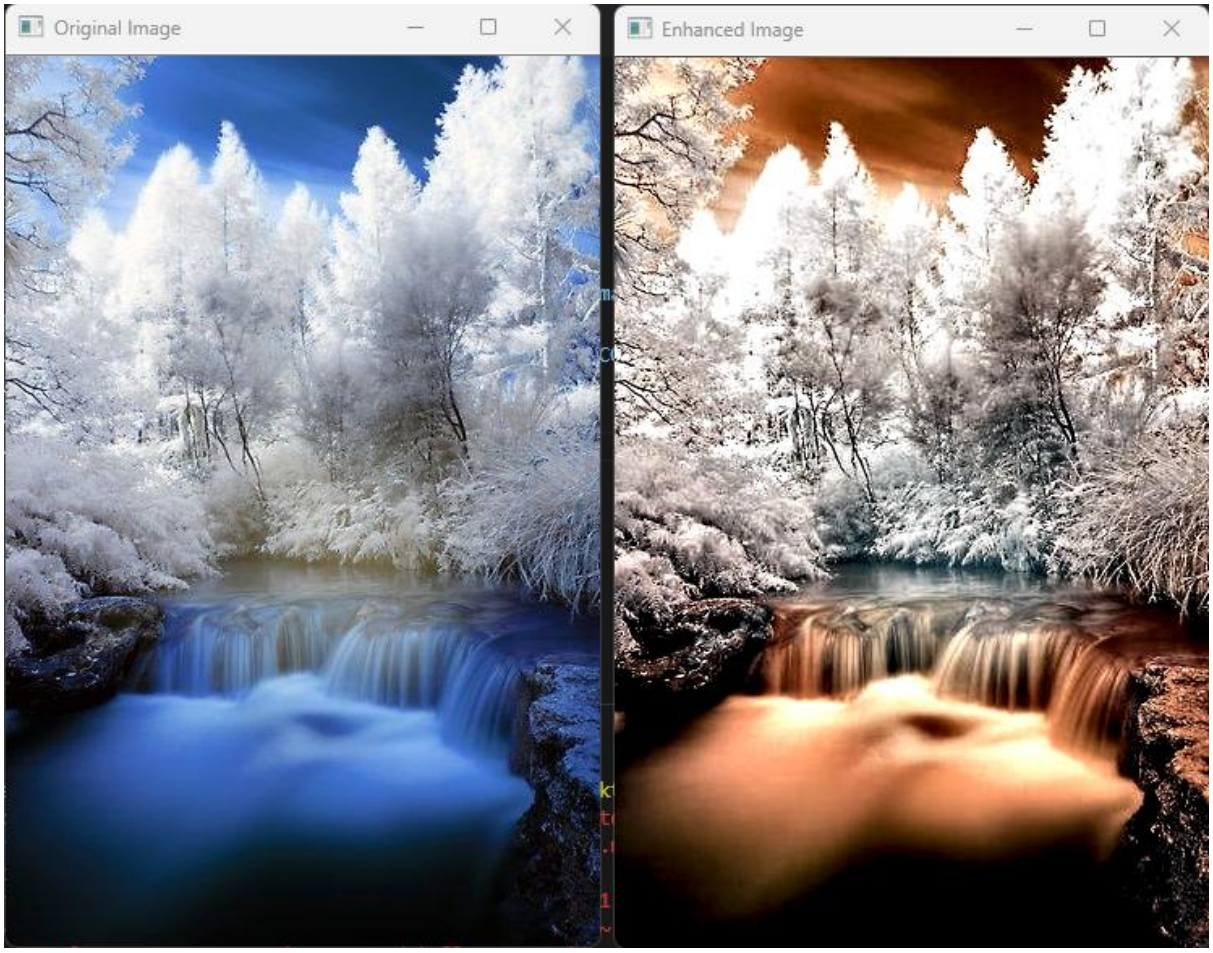
Algorithm:

1. Read the Image.
2. Convert color from BGR to RGB.
3. Enhance the color of the image
4. Display the image.

Python code:

```
import cv2
img = cv2.imread('sharp.png')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# Enhance the color of the image
enhanced  = cv2.detailEnhance(img, sigma_s=100, sigma_r= 0.15)
cv2.imshow('Original Image', cv2.cvtColor(img, cv2.COLOR_RGB2BGR))
cv2.imshow('Enhanced Image',enhanced)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Outputs:



Color Enhancement using **cv2.detailEnhance()** function which enhance the details and colors in the image. The **sigma_s** and **sigma_r** parameters control the size of the details to enhance and the strength of the enhancement, respectively.

Experiment no: 05

Experiment name To create an opencv-python program to enhance brightness of a given RGB/grayscale image.

Theory:

Brightness: Brightness is the measured intensity (amount of light) of all pixels. Enhancing brightness refers to a set of techniques that are used to increase the overall brightness of an image. Histogram Equalization can be used for enhance brightness.

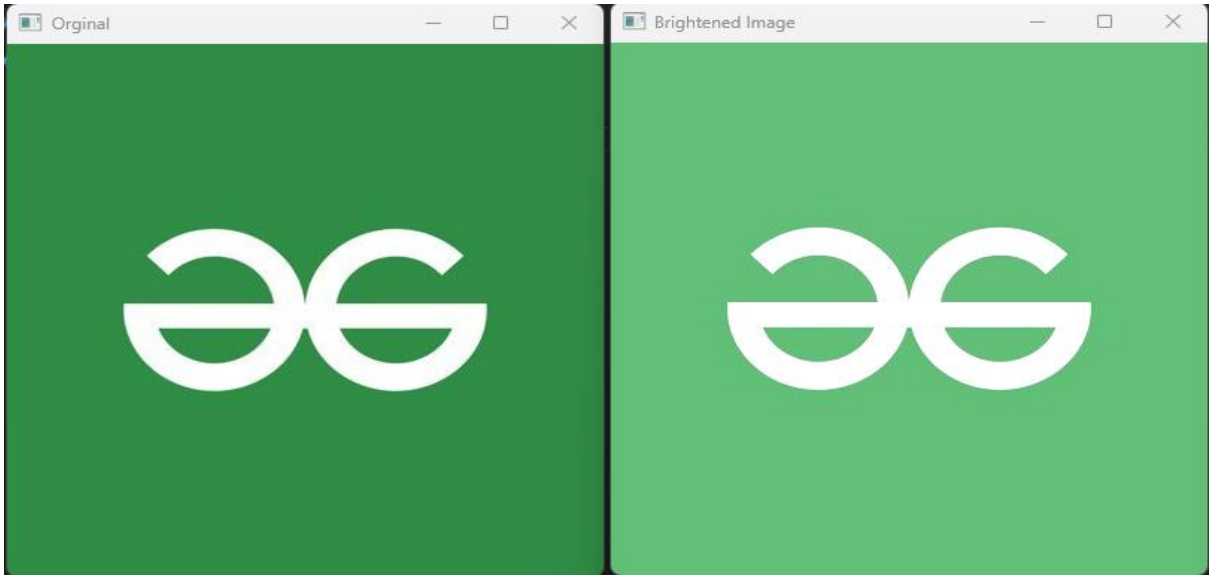
Algorithm:

1. Read the Image.
2. Split the Image into b, g, r.
3. Create a matrix which each coordinate value is 1.
4. Add intensity within b, g, r.
5. Merge the image.
6. Display the Image.

Python code:

```
import cv2
import numpy as np
img = cv2.imread("gfg.png")
b, g,r = cv2.split(img)
m = img.shape[0]
n = img.shape[1]
matrix = np.ones((m,n), dtype="uint8")
# increase the intensity of each color channel
b = cv2.add(b, matrix*50)
g = cv2.add(g, matrix*50)
r = cv2.add(r, matrix*50)
brightened_img = cv2.merge([b, g, r])
cv2.imshow("Original", img)
cv2.imshow("Brightened Image", brightened_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

outputs:



Experiment no: 06

Experiment name: To create an opencv-python program to enhance contrast of a given RGB/grayscale image.

Theory:

Image contrast: Contrast can be described as the particular amount of light that is emitted from a particular object in the real world, or in a digital image. The contrast of an image is increased, the light portions of the image will become lighter, and the dark portions, darker.

Algorithm:

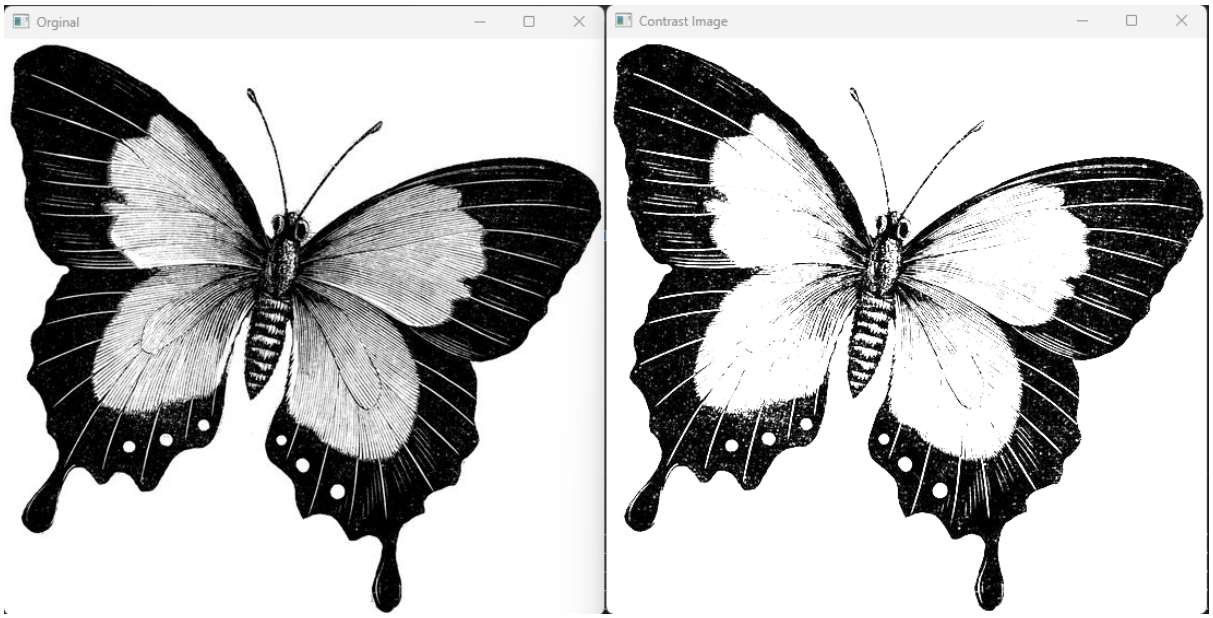
1. Import the required library **OpenCV.**
2. Read the input image using **cv2.imread()** method. Specify the full path of the image.

3. Define **alpha** (it controls contrast) and **beta** (it controls brightness) and call **convertScaleAbs()** function to change the contrast and brightness of the image. This function returns the image with adjusted contrast and brightness. Alternatively, we can also use the **cv2.addWeighted()** method to change contrast and brightness
4. Display the contrast image.

Python code:

```
import cv2
image = cv2.imread('b.jpg')
alpha = 2.5 # Contrast control
beta = 0 # Brightness control
adjusted = cv2.convertScaleAbs(image, alpha=alpha, beta = beta)
cv2.imshow("Original", image)
cv2.imshow('Contrast Image', adjusted)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Outout:



Experiment no: 07

Experiment name: To create an opencv-python program to segment a given RGB/grayscale image based on thresholding.

Theory:

Segmentation: Image processing and computer vision, image segmentation refers to the process of dividing an input image into multiple segments or regions, each of which corresponds to a particular object or part of the image. The goal of image segmentation is to simplify or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is an important pre-processing step for many computer vision applications, such as object detection, object tracking, image recognition, and image analysis.

Threshold: Thresholding is a technique in OpenCV, which is the assignment of pixel values in relation to the threshold value provided. In thresholding, each pixel value is compared with the threshold value. If the pixel value is smaller than the threshold, it is set to 0, otherwise, it is set to a maximum value (generally 255).

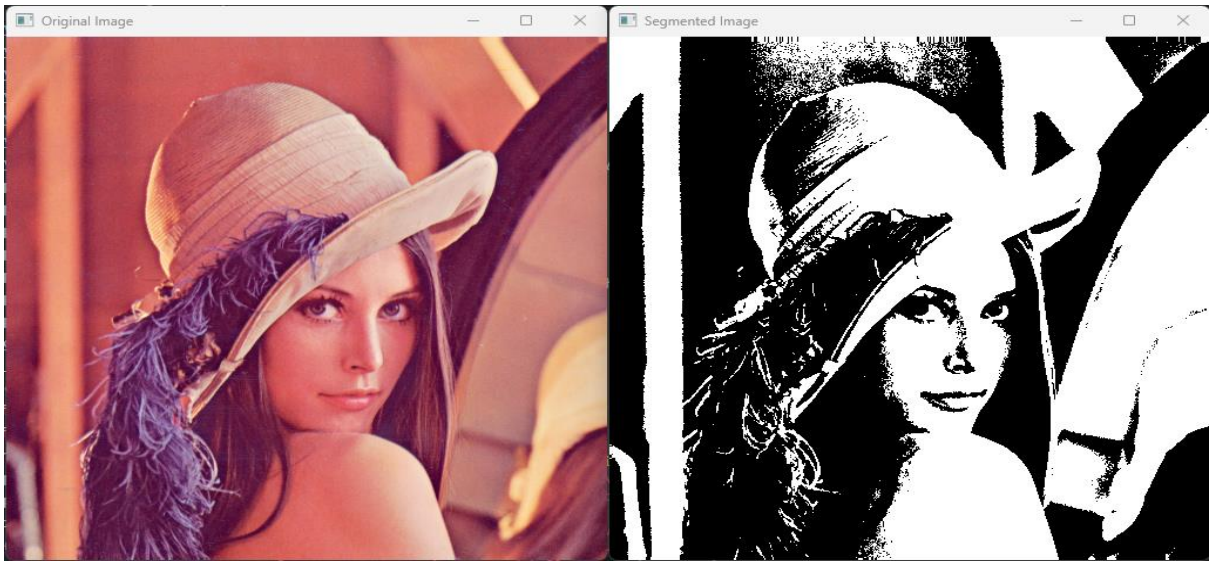
Algorithm:

1. Read the Image.
2. Convert the color from BGR to RGB.
3. Define a specific threshold value.
4. Calculated threshold on the image.
5. Display the Image.

Python code:

```
import cv2
img = cv2.imread('Lenna.png')
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Apply binary thresholding to the image
thresh_val = 128
max_val = 255
thresh_type = cv2.THRESH_BINARY
_, binary_img = cv2.threshold(gray_img, thresh_val, max_val, thresh_type)
cv2.imshow("Original Image", img)
cv2.imshow("Segmented Image", binary_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Outputs:



Experiment no: 08

Experiment Name: To create an opencv-python program to segment a given RGB/grayscale image based on the contour detection algorithm.

Contour detection: It is a technique in computer vision and image processing that involves finding the curves or boundaries of objects in an image. It is used to extract and represent the shape of objects in an image

Application of Contours in Computer Vision:

- **Motion Detection**.
- **Background / Foreground Segmentation:** To replace the background of an image with another, you need to perform image-foreground extraction (similar to image segmentation).

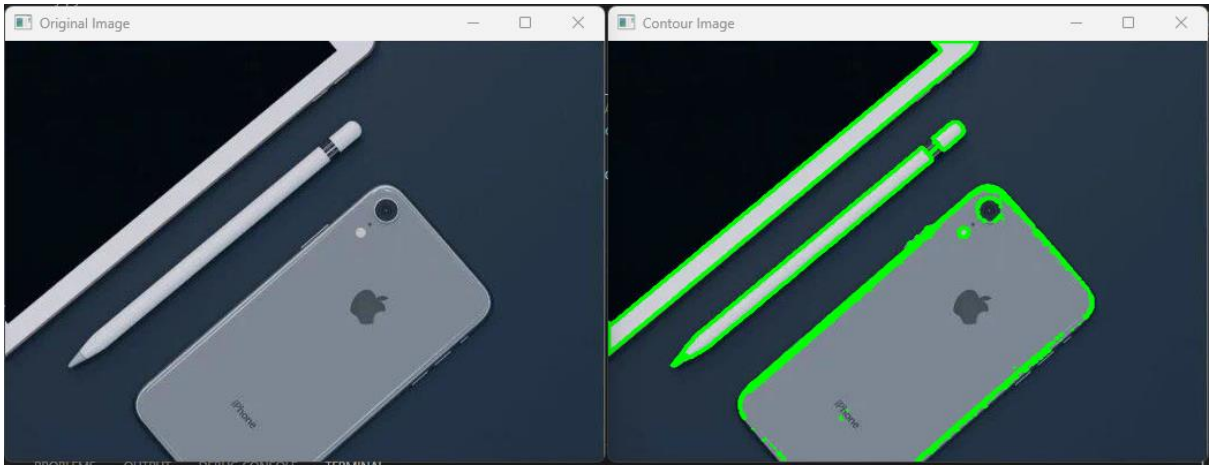
Algorithm:

1. Read the Image and convert it to Grayscale Format
2. Apply Binary Thresholding
3. Find the Contours
4. Draw Contours on the Original RGB Image.
5. Display the Image.

Python code:

```
# Experiment no: 08
import cv2
image = cv2.imread('contour.jpg')
img_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(img_gray, 150, 255, cv2.THRESH_BINARY)
# detect the contours on the binary image using cv2.CHAIN_APPROX_NONE
contours, hierarchy = cv2.findContours(image=thresh, mode=cv2.RETR_TREE,
method=cv2.CHAIN_APPROX_NONE)
image_copy = image.copy()
cv2.drawContours(image=image_copy, contours=contours, contourIdx=-1, color=(0,
255, 0), thickness=2, lineType=cv2.LINE_AA)
cv2.imshow("Original Image", image)
cv2.imshow('Contour Image', image_copy)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Outputs:



Code Explanation given below:

1. Start with the **findContours()** function. It has three required arguments, as shown below.
    a. **image**: The binary input image obtained in the previous step.
    b. **mode**: This is the contour-retrieval mode. We provided this as **RETR_TREE,** which means the algorithm will retrieve all possible contours from the binary image.
    c. method: This defines the contour-approximation method. We will use **CHAIN_APPROX_NONE**. Though slightly slower than **CHAIN_APPROX_SIMPLE**, we use this method here to store ALL contour points.
2. **drawContours()** function to overlay the contours on the RGB image. This function has four required arguments.
    a. **image**: This is the input RGB image on which you want to draw the contour.
    b. **contours**: The list of contours to draw.
    c. **contourIdx**: Index of the contour to draw. If it is negative, all the contours are drawn.
    d. **color**: This indicates the color of the contour points you want to draw. We are drawing the points in green.
    e. **thickness**: This is the thickness of contour points.
    f. **lineType:** The type of line used for drawing the contour.

Experiment no: 09

Experiment Name: To create an opencv-python program to segment a given RGB/grayscale image based on K-means algorithm.

Theory:

K-means algorithm: In digital image processing, the K-means algorithm is often used for image segmentation. The algorithm can be used to partition an image into K clusters, where each pixel in the image is assigned to the cluster with the nearest mean or centroid. This process

is often used to segment an image into regions or objects that have similar characteristics or properties.

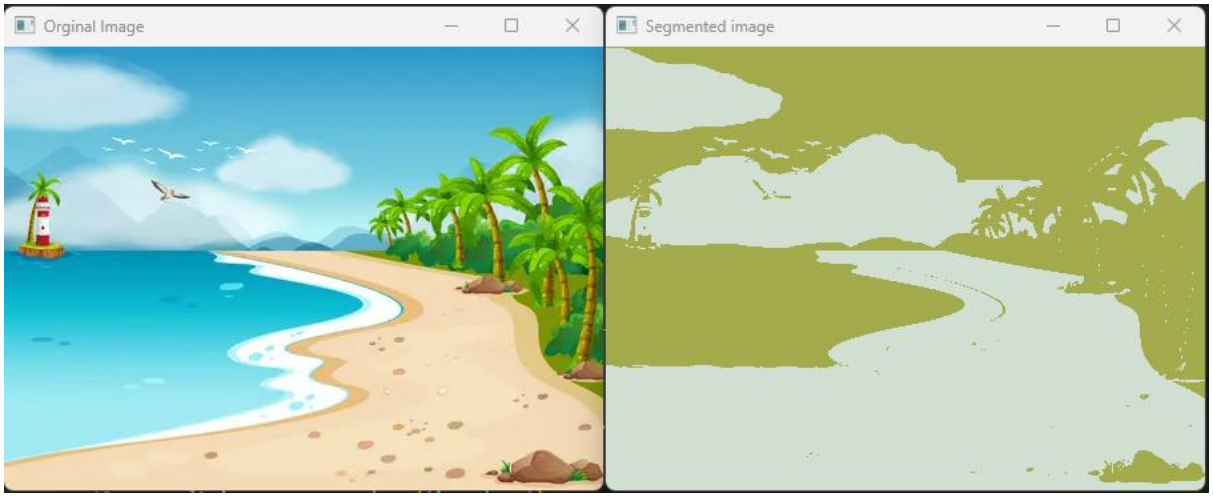
Algorithm:

1. Choose the number of clusters K and randomly initialize K cluster centroids.
2. Assign each pixel in the image to the closest cluster centroid based on its color value.
3. Recalculate the centroids of each cluster as the mean color of all the pixels assigned to that cluster.
4. Repeat steps 2-3 until convergence or a maximum number of iterations is reached.
5. Output the final segmented image where each pixel is labeled with the cluster it belongs to.

Python code:

```
# Experiment no: 09
import numpy as np
import cv2
path = 'itsohk.png'
image = cv2.imread(path, -1)
# Preprocessing the Image
img = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
twoDimage = img.reshape((-1,3))
twoDimage = np.float32(twoDimage)
# Defining Parameters
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
K = 2
attempts=10
# Apply K-Means
ret,label,center=cv2.kmeans(twoDimage,K,None,criteria,attempts,cv2.KMEANS_PP_C
ENTERS)
center = np.uint8(center)
res = center[label.flatten()]
result_image = res.reshape((img.shape))
cv2.imshow("Orginal Image", image)
cv2.imshow('Segmented image', result_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Outputs:



Opencv function explanation given below:

1. Firstly, load the image using the **cv2.imread()** function.
2. reshape the image to a 2D array of pixels and 3 color channels (assuming it's an RGB image). also convert the pixel values to the float32 data type, which is required by the k-means algorithm.
3. (**cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0**) is a tuple used to define the criteria for convergence in k-means clustering.
    a. cv2.TERM_CRITERIA_EPS and cv2.TERM_CRITERIA_MAX_ITER are two flags used to indicate the criteria for convergence.
    b. cv2.TERM_CRITERIA_EPS specifies the required accuracy of the cluster centers. The algorithm stops when the relative change of the center positions is less than 1.0.
    c. cv2.TERM_CRITERIA_MAX_ITER specifies the maximum number of iterations allowed for the algorithm.
4. Apply the k-means algorithm using the **cv2.kmeans()** function. The first argument is the pixel values, the second argument is the number of clusters, the third argument is the initial cluster centers (which we set to None), the fourth argument is the criteria for convergence, the fifth argument is the number of attempts to run the algorithm with different initializations, and the last argument is the flag to indicate whether to use random or specific initial centers for clustering.

Experiment no: 10

Experiment Name: To create an opencv-python program to sharp a given RGB/grayscale image.

Theory:

Sharpening: Highlight fine detail in an image and enhance detail that has been blurred.

We will now look at the process of sharpening an image, we will make use of a kernel to highlight each particular pixel and enhance the color that it emits.

A kernel is known by other names such as:

1. Convolution Matrix.
2. Mask.
3. Matrix/Array.

Convolution- i.e, the process during which the kernel is applied to the image. There is a fixed/standard general formula for convolutions (blurring, sharpening, etc). This formula is as follows:

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} \omega(s,t) f(x-s, y-t)$$

$$g = \omega * f$$

To sharpen an image in Python, we are required to make use of the **filter2D()** method.
Syntax: filter2D (src, ddepth, kernel)
Parameters:

- **Src –** The source image to apply the filter on.
- **Ddepth –** Depth of the output image [-1 will give the output image depth as same as the input image]
- **Kernel –** The 2d matrix we want the image to convolve with.

Since we wish to sharpen an image our kernel will be as follows:

Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$

Algorithm:

1. Read the Image.
2. Take a 3x3 kernel.
3. Use filter2D for sharping the image.
4. Display the image.

Python code:

```
import cv2
import numpy as np
image = cv2.imread('sharp.png', flags=cv2.IMREAD_COLOR)
kernel = np.array([[0, -1, 0],
                   [-1, 5,-1],
                   [0, -1, 0]])
image_sharp = cv2.filter2D(src=image, ddepth=-1, kernel=kernel)
cv2.imshow("Orginal Image", image)
cv2.imshow('Sharpened Image', image_sharp)
cv2.waitKey()
cv2.destroyAllWindows()
```

Outputs:

Experiment no: 11

Experiment Name: To create an opencv-python program to skeletonize a given RGB/grayscale image.

Theory:

Skeletonization: Skeletonization is a process in image processing that involves reducing the shape of an object to its "skeleton" or "centerline" while preserving its topology or connectivity. The skeleton of an object is its thinnest representation that can be used to describe its overall shape and structure.

Applications such as shape analysis, pattern recognition, and object tracking.

Algorithm:

1. Read the image.
2. Resize the image.
3. Apply binary threshold on the image
4. Determine skeletonize on the image
5. Take a 3x3 kernel
6. Dilate the skeletonize image

7. Applied median filter on the image
8. Applied inverse binary threshold on the output image.
9. Display the image.

Python code:

```
import numpy as np
import cv2
img = cv2.imread('Thumb.png',0)
img = cv2.resize(img, (580,580))
ret,img = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
# Function for skeletonizing the image
def findSkeleton(im):
    element = cv2.getStructuringElement(cv2.MORPH_CROSS,(3,3))
    out = np.zeros(im.shape,np.uint8)
    flag = 0
    while(not flag):
        eroded = cv2.erode(im, element)
        opened = cv2.dilate(eroded, element)
        opened = cv2.subtract(im,opened)
        out = cv2.bitwise_or(out,opened)
        im = eroded.copy()
        zeros = img.size - cv2.countNonZero(im)
        flag = 1 if (zeros == img.size) else 0
    return out
output = findSkeleton(img)
kernel = np.ones((3,3),np.uint8)
output = cv2.dilate(output,kernel)
output = cv2.medianBlur(output, 5)
ret,thresh = cv2.threshold(output,127,255,cv2.THRESH_BINARY_INV)
res = np.hstack((img, thresh))
cv2.imshow("output", cv2.resize(res, dsize=None,fx=0.5, fy=0.5))
cv2.imwrite("task1_output.png", res)
cv2.waitKey(0)
```

Outputs:

Here **cv2.erode(im, element)** is a function in the OpenCV library for Python that performs morphological erosion on an image **im** using a structuring element specified by **element**.

Morphological erosion is a fundamental operation in image processing that involves shrinking the boundaries of objects in an image. It is used to remove small details, smooth contours, and separate objects that are too close to each other.

Experiment no: 12

Experiment Name: To create an opencv-python program to smooth a given RGB/grayscale image by a 2Dfilter.

Theory:

Removal of small details from an image prior to object extraction.

Smoothing: Used for blurring and for noise reduction.

1. Blurring is used in preprocessing steps, such as
    - Removal of small details from an image prior to object extraction.
    - bridging of small gaps in lines or curves
2. Noise reduction can be accomplished by blurring with a linear filter and also by a nonlinear filter

2D filters can be used to smooth an image, which can help to reduce the amount of high-frequency content in the image.

Algorithm:

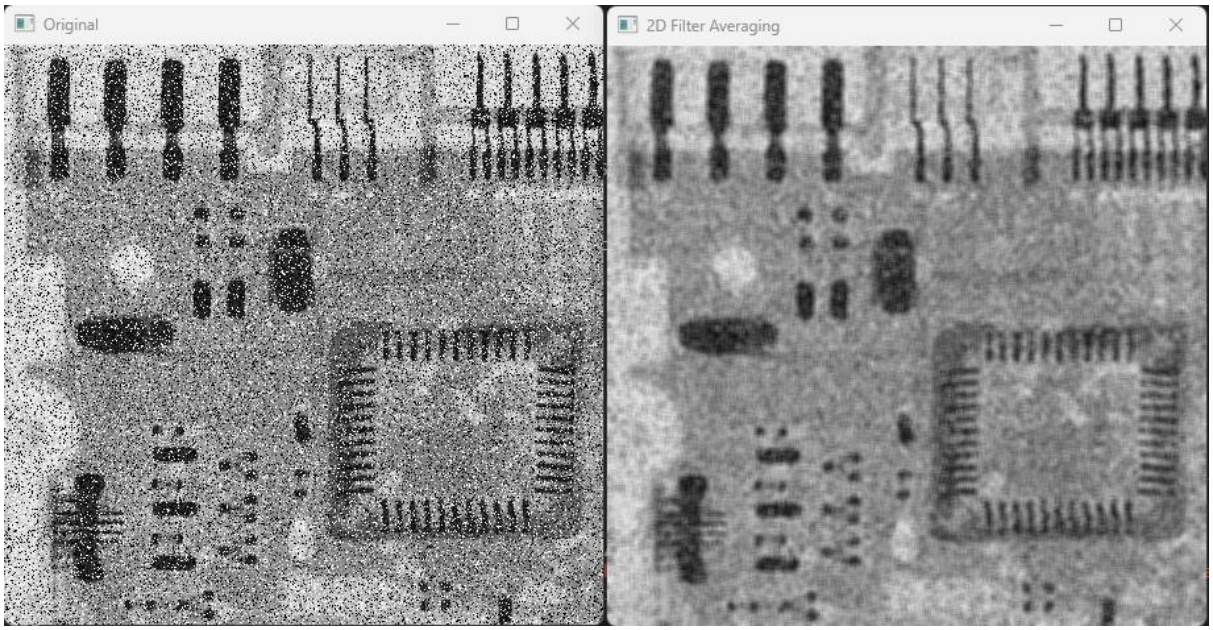
1. Read the Image.
2. Resize the Image.
3. Create a 5x5 kernel Box Filter which values are one.

4. Use 2D filter for smooth the image.
5. Display the Image.

Python code:

```
# Experiment no:12
import cv2
import numpy as np
img = cv2.imread('inp3.tif')
kernel = np.ones((5,5),np.float32)/25
dst = cv2.filter2D(img,-1,kernel)
cv2.imshow("Original", img)
cv2.imshow("2D Filter Averaging", dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Outputs:



Experiment no: 13

Experiment Name: To create an opencv-python program to smooth a given RGB/grayscale image by an average filter.

Theory:

Average filter: Output is simply the average of the pixels contained in the neighborhood of the filter mask called averaging filters or low-pass filters.

Box filter: The center pixel and neighbor pixels are equally important.

$$1/9 \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Weighted Average Filter: The center is the most important and other pixels are inversely weighted as a function of their distance from the center of the mask.

$$\frac{1}{16} \times \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Algorithm:

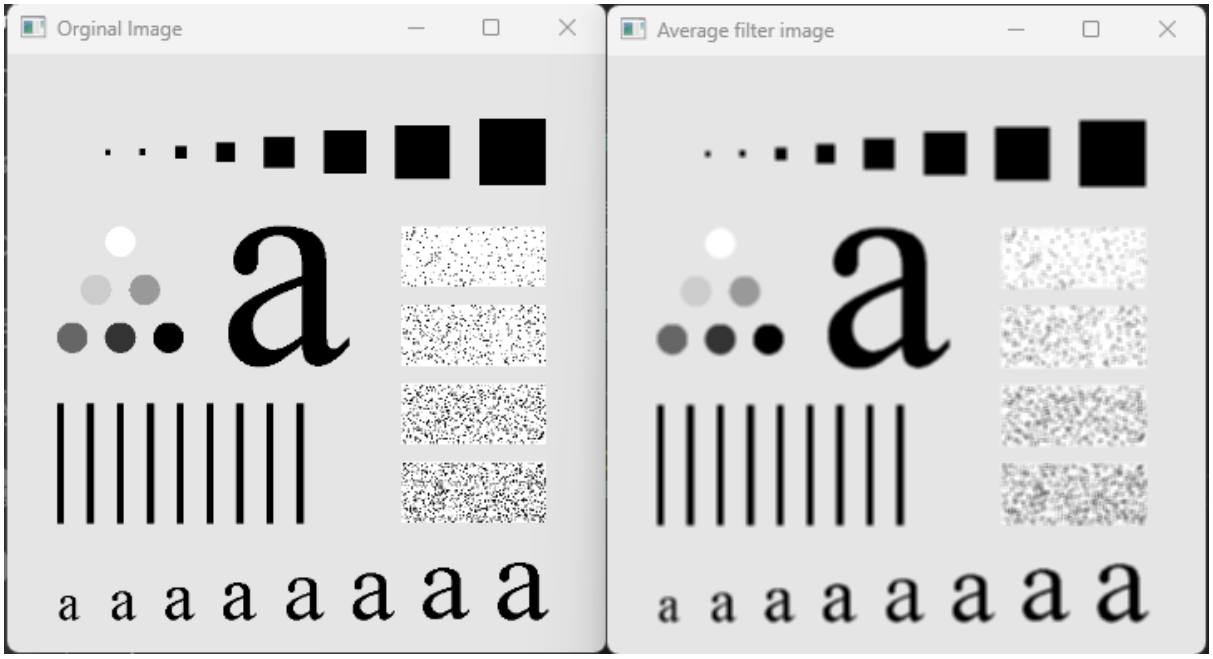
1. Read the Image.
2. Resize the Image.
3. Create a 3x3 kernel Box Filter which values are one.
4. Use 2D filter for smooth the image.
5. Display the Image.

Python code:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('inp1.tif')
img = cv2.resize(img, (360,360))
kernel = np.ones((3,3),np.float32)/9
blur = cv2.filter2D(img,-1,kernel)
#blur = cv2.blur(img,(5,5))
#image show
cv2.imshow("Orginal Image", img)
cv2.imshow("Average filter image", blur)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Outputs:



Experiment no: 15

Experiment Name: To create an opencv-python program to smooth a given RGB/grayscale image by a Gaussian filter.

Theory:

Gaussian Filter: A Gaussian Filter is a low pass filter used for reducing noise (high frequency components) and blurring regions of an image. The filter is implemented as an Odd sized Symmetric Kernel which is passed through each pixel of the Region of Interest to get the desired effect. Gaussian function:

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

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

Gaussian kernel

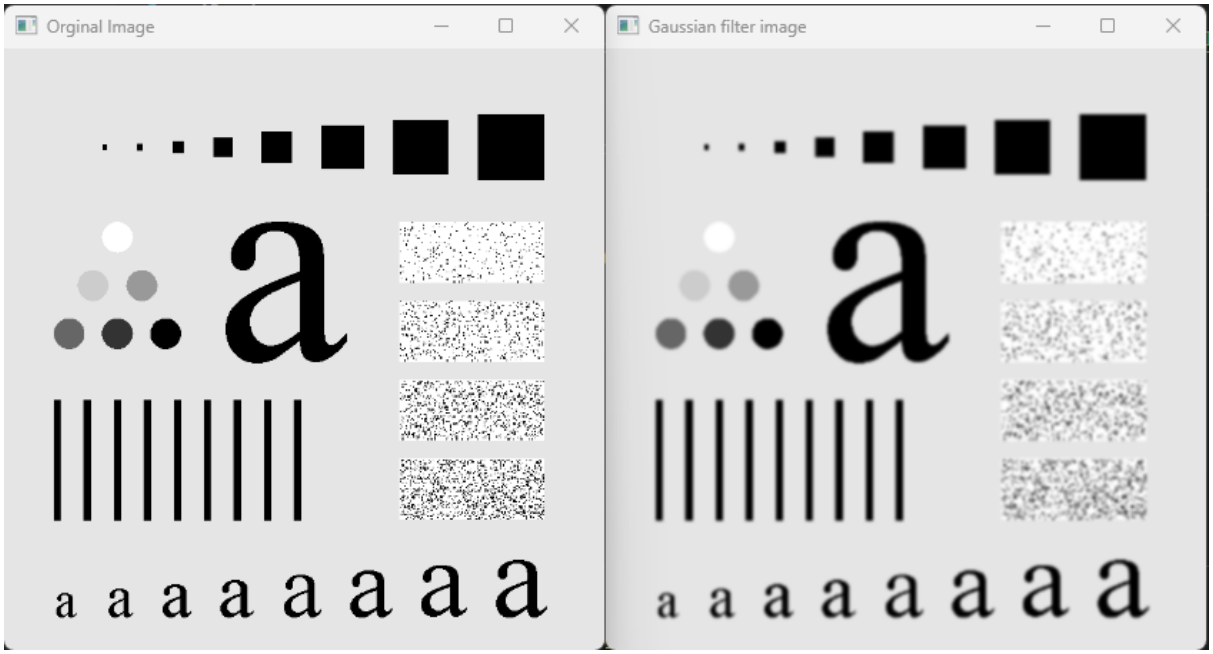
Algorithm:

1. Read the Image.
2. Resize the Image.
3. Use Gaussian filter for smooth the image.
4. Display the Image.

Python code:

```python
# Experiment no: 15
import cv2
import numpy as np
img = cv2.imread('inp1.tif')
img = cv2.resize(img, (420,420))
blur = cv2.GaussianBlur(img,(5,5),0)
cv2.imshow("Orginal Image", img)
cv2.imshow("Gaussian filter image", blur)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Outputs:



Experiment no: 16

Experiment Name: To create an opencv-python program to smooth a given RGB/grayscale image by a median filter.

Theory:

Median filter: Replaces the value of a pixel by the median of the gray levels in the neighborhood of that pixel

- Quite popular because for certain types of random noise such as impulse noise(or salt and pepper noise), they provide excellent noise-reduction capabilities, with considering less blurring than linear smoothing filters of similar size.

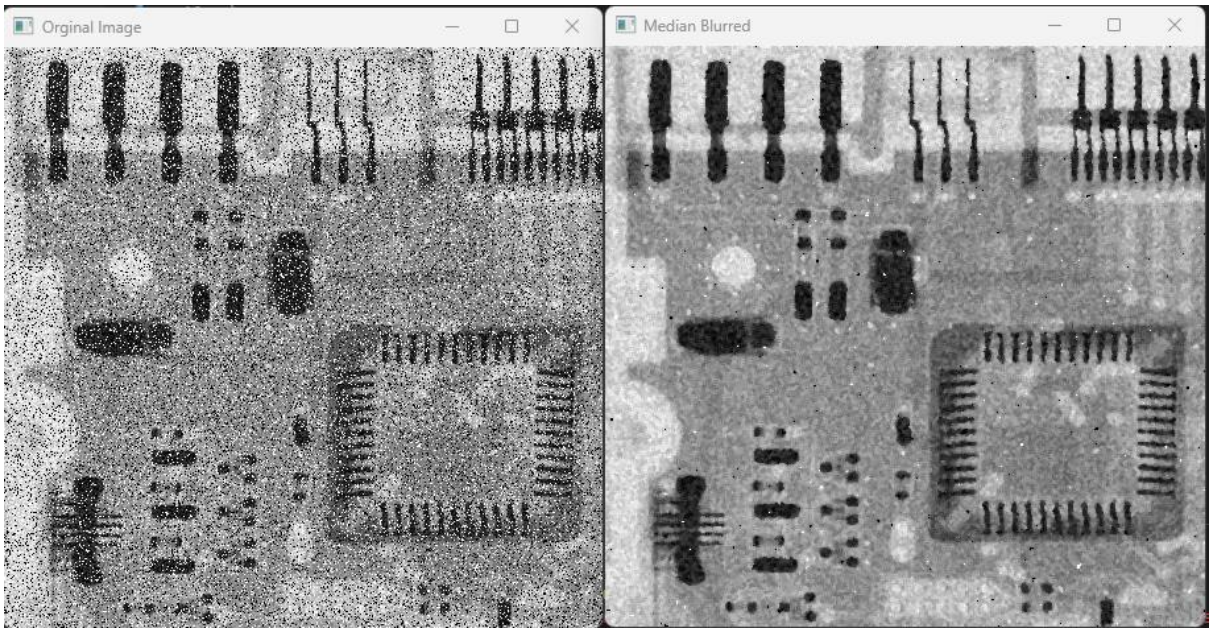
Algorithm:

1. Read the Image.
2. Resize the Image.
3. Use median filter for smooth the image.
4. Display the Image.

Python code:

```
import cv2
img = cv2.imread('inp3.tif')
blur = cv2.medianBlur(img, 3)
#image show
cv2.imshow("Orginal Image", img)
cv2.imshow("Median Blurred", blur)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Outputs:



Experiment no: 17

Experiment Name: To create an opencv-python program to smooth a given RGB/grayscale image by a bilateral filter.

Theory:

Bilateral filter: Bilateral filter is highly effective in noise removal while keeping edges sharp. But the operation is slower compared to other filters. We already saw that a Gaussian filter takes the neighbourhood around the pixel and finds its Gaussian weighted average. This Gaussian filter is a function of space alone, that is, nearby pixels are considered while filtering. It doesn't consider whether pixels have almost the same intensity. It doesn't consider whether a pixel is an edge pixel or not. So it blurs the edges also, which we don't want to do.

Bilateral filtering also takes a Gaussian filter in space, but one more Gaussian filter which is a function of pixel difference. The Gaussian function of space makes sure that only nearby pixels are considered for blurring, while the Gaussian function of intensity difference makes sure that only those pixels with similar intensities to the central pixel are considered for blurring. So it preserves the edges since pixels at edges will have large intensity variation.

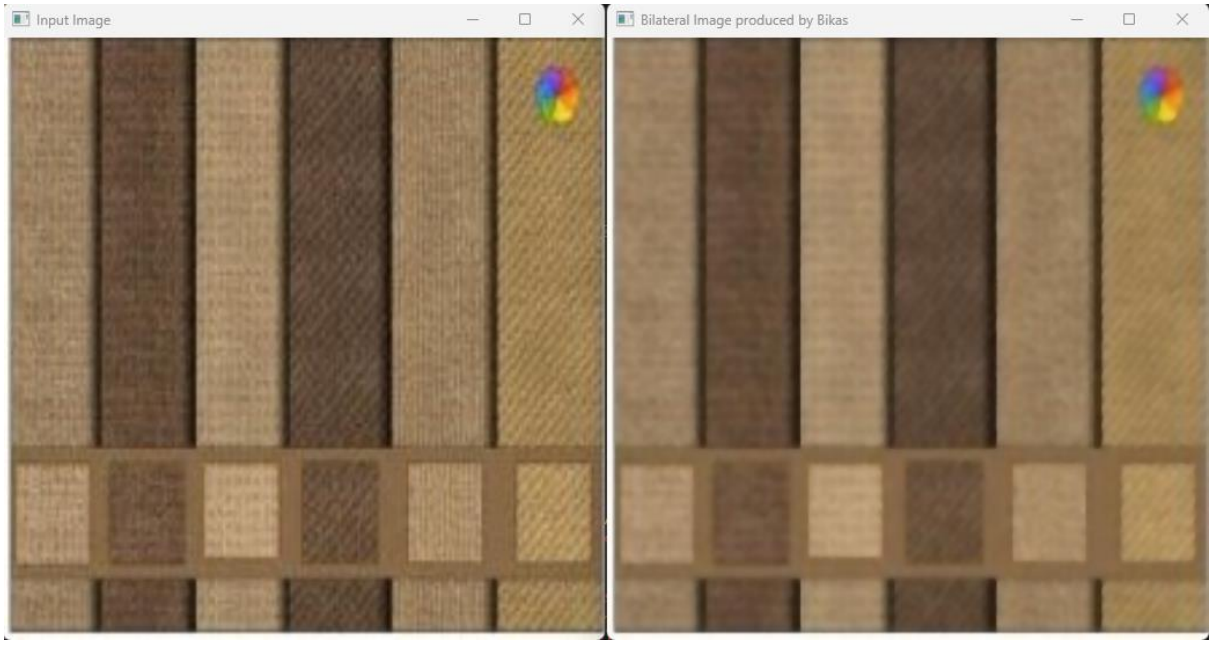
Algorithm:

1. Read the Image.
2. Resize the Image.
3. Use bilateral filter for smooth the image.
4. Display the Image.

Python code:

```
import cv2
# Load the input image
img = cv2.imread('bilateral.jpg')
img = cv2.resize(img, (520,520))
# Apply bilateral filter to the image
bilateral = cv2.bilateralFilter(img, 9, 75, 75)
# Display the original and filtered images
cv2.imshow('Input Image', img)
cv2.imshow('Filtered Image', bilateral)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Outputs:



In this program, a bilateral filter to the image using the **cv2.bilateralFilter()** function with a kernel size of 9, a sigma value of 75 for the color space, and a sigma value of 75 for the coordinate space.