METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

Estd:2008

(Affiliated to Osmania University & Approved by AICTE, New Delhi)

LABORATORY MANUAL

DIGITAL IMAGE PROCESSING LAB

BE V Semester (AICTE Model Curriculum): 2023-24

NAME: _____

ROLL NO:_____

BRANCH:_____ **SEM:**_____

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERNG

Empower youth- Architects of Future World

VISION

To produce ethical, socially conscious and innovative professionals who would contribute to sustainable technological development of the society.

MISSION

To impart quality engineering education with latest technological developments and interdisciplinary skills to make students succeed in professional practice.

To encourage research culture among faculty and students by establishing state of art laboratories and exposing them to modern industrial and organizational practices.

To inculcate humane qualities like environmental consciousness, leadership, social values, professional ethics and engage in independent and lifelong learning for sustainable contribution to the society.

METHODIST

Estd:2008 COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT
OF
COMPUTER SCIENCE AND ENGINEERING

LABORATORY MANUAL

DIGITAL IMAGE PROCESSING LAB

Prepared
By

UNNATI KHANAPURKAR,

Assistant Professor

&

A.A.R.SENTHIL KUMAAR

Assistant Professor.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION & MISSION

VISION

To become a leader in providing Computer Science & Engineering education with emphasis on knowledge and innovation.

MISSION

- To offer flexible programs of study with collaborations to suit industry needs.
- To provide quality education and training through novel pedagogical practices.
- To expedite high performance of excellence in teaching, research and innovations.
- To impart moral, ethical values and education with social responsibility.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM EDUCATIONAL OBJECTIVES

After 3-5 years of graduation, the graduates will be able to

PEO1: Apply technical concepts, Analyze, Synthesize data to Design and create novel products and solutions for the real life problems.

PEO2: Apply the knowledge of Computer Science Engineering to pursue higher education with due consideration to environment and society.

PEO3: Promote collaborative learning and spirit of team work through multidisciplinary projects

PEO4: Engage in life-long learning and develop entrepreneurial skills.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM OUTCOMES

Engineering graduates will be able to:

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the Engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES

At the end of 4 years, Computer Science and Engineering graduates at MCET will be able to:

PSO1: Apply the knowledge of Computer Science and Engineering in various domains like networking and data mining to manage projects in multidisciplinary environments.

PSO2: Develop software applications with open-ended programming environments**.**

PSO3: Design and develop solutions by following standard software engineering principles and implement by using suitable programming languages and platforms

DIGITAL IMAGE PROCESSING LAB

Semester V	L	T	P	Credits
Subject code – 3ES551CS	0	0	2	1

Course Objectives:	Course Outcomes:
➢ To introduce the concepts of image processing and basic analytical methods to be used in image processing. ➢ To familiarize students with image enhancement and restoration techniques, ➢ To explain different image compression techniques. ➢ To introduce segmentation and morphological processing techniques.	1. Understand how the images are read as grayscale and RGB 2. Understand how the images are getting converted in different forms 3. Understand the processing and implement different image filtering techniques 4. Implement Edge detection 5. Compare the different DFT, DCT and DWT techniques

List of Programs:

1. OpenCV installation
2. Reading, Writing and Storing Images
3. Reading an Image as Grayscale
4. Reading Image as RGB
5. Image Conversion - Colored Images to GrayScale
6. Image Conversion - Colored Image to Binary
7. Processing – Blur – Averaging, Gaussian
8. Image Filtering - Bilateral Filter, Box Filter, Erosion
9. Thresholding – Simple, Adaptive
10. Sobel Operator
11. DFT, DCT, DWT
12. Edge Detection

Head of the Department
Department of CSE
Methodist College of Engg & Tech
Abids, Hyderabad.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Course Outcomes (CO's):

SUBJECT NAME: DIGITAL IMAGE PROCESSING CODE: 3ES501CS

SEMESTER: V

CO No.	Course Outcomes	Taxonomy Level
PC253CS.1	Explain the fundamentals of digital image and its processing	Understanding
PC253CS.2	Understand the enhancement, segmentation, restoration, compression processes on an image.	Understanding
PC253CS.3	Explore the fundamental relation between pixels and apply image enhancement, filtering techniques, morphological operations for image processing.	Apply
PC253CS.4	Explore the utility of 2-D transforms and analyze the different linear image restoration techniques.	Analyze
PC253CS.5	Evaluate point processing techniques, histogram manipulation, compression techniques and mathematical model for image restoration	Evaluate

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the program / experiment details.
3. Student should enter into the laboratory with:
    a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
    b. Laboratory Record updated up to the last session experiments.
    c. Formal dress code and Identity card.
4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviours with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out. If anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Head of the Department **Principal**

CODE OF CONDUCT FOR THE LABORATORY

- All students must observe the dress code while in the laboratory
- Footwear is NOT allowed
- Foods, drinks and smoking are NOT allowed
- All bags must be left at the indicated place
- The lab timetable must be strictly followed
- Be PUNCTUAL for your laboratory session
- All programs must be completed within the given time
- Noise must be kept to a minimum
- Workspace must be kept clean and tidy at all time
- All students are liable for any damage to system due to their own negligence
- Students are strictly PROHIBITED from taking out any items from the laboratory
- Report immediately to the lab programmer if any damages to equipment

BEFORE LEAVING LAB:

- Arrange all the equipment and chairs properly.
- Turn off / shut down the systems before leaving.
- Please check the laboratory notice board regularly for updates.

Lab In – charge

METHODIST

Estd:2008 COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LIST OF EXPERIMENTS

S.No.	Name of the Experiment	Date of Experiment	Date of Submission	Page No	Faculty Signature
1.	Simulation and Display of an Image, Negative of an Image (Binary Grayscale)				
2.	Implementation of Relationships between Pixels				
3.	Implementation of Transformations of an Image				
4.	Contrast stretching of allow contrast image,Histogram,and Histogram Equalization				
5.	Display of bit planes of an Image				
6.	Display of FFT(1-D& 2-D) of an image				
7.	Computation of Mean, Standard Deviation, Correlation coefficient of the given Image.				
8.	Implementation of Image Smoothening Filters(Mean and Median filtering of an Image)				
9.	Implementation of image sharpening filters and Edge Detection using Gradient Filters				
10.	Image Compression by DCT,DPCM,HUFFMAN coding				
11.	Implementation of image restoring techniques				
12.	Implementation of Image Intensity slicing technique for image enhancement				
13.	Canny edge detection Algorithm				

ADDITIONAL PROGRAMS

S.No.	Name of the Experiment	Date of Experiment	Date of Submission	Page No	Faculty Signature
1	Perform Image Translation Transformation using OpenCV				
2	Perform Image Scaling Transformation using OpenCV				
3	Perform Image Reflection Transformation using OpenCV				
4	Perform Image Rotation Transformation using OpenCV				
5	Perform Image Crop Transformation using OpenCV				
6	Perform Image Shearing Transformation using OpenCV				

PROGRAM 1: Open CV Installation

Aim : Write the steps to install Open CV

Computer vision is a field of computer science that focuses on enabling computers to identify and understand objects and people in images and videos. Like other types of AI, computer vision seeks to perform and automate tasks that replicate human capabilities.

OpenCV is an open-source library for the computer vision. It provides the facility to the machine to recognize the faces or objects.

- OpenCV is a Python open-source library, which is used for computer vision in Artificial intelligence, Machine Learning, face recognition, etc.
- In OpenCV, the **CV is an abbreviation form of a computer vision**, which is defined as a field of study that helps computers to understand the content of the digital images such as photographs and videos.
- The purpose of computer vision is to understand the **content of the images**. It extracts the description from the pictures, which may be an object, a text description, and three-dimension model, and so on. For example, cars can be facilitated with computer vision, which will be able to identify and different objects around the road, such as traffic lights, pedestrians, traffic signs, and so on, and acts accordingly.

Computer vision allows the computer to perform the same kind of tasks as humans with the same efficiency. There are a two main task which are defined below:

- **Object Classification** - In the object classification, we train a model on a dataset of particular objects, and the model classifies new objects as belonging to one or more of your training categories.
- **Object Identification** - In the object identification, our model will identify a particular instance of an object - for example, parsing two faces in an image and tagging one as Virat Kohli and other one as Rohit Sharma.

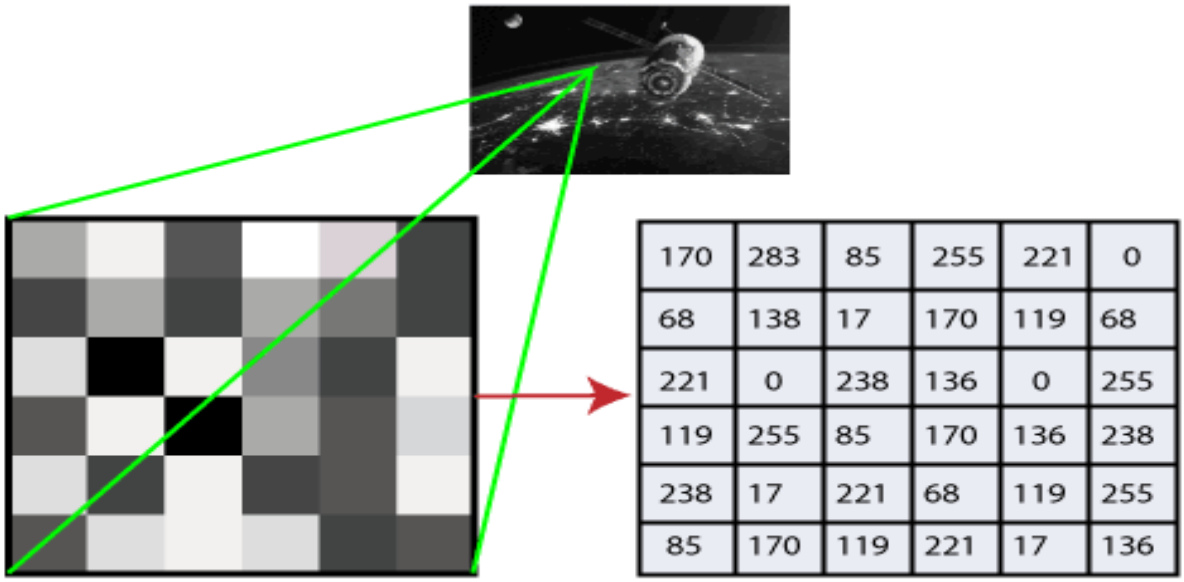
History

• **OpenCV stands for Open Source Computer Vision Library**, which is widely used for image recognition or identification.
• It was officially launched in 1999 by Intel.
• It was written in C/C++ in the early stage, but now it is commonly used in Python for the computer vision as well.
• The first alpha version of OpenCV was released for the common use at the IEEE Conference on Computer Vision and Pattern Recognition in 2000, and between 2001 and 2005, five betas were released. The first 1.0 version was released in 2006.
• The second version of the OpenCV was released in October 2009 with the significant changes. The second version contains a major change to the C++ interface, aiming at easier, more type-safe, pattern, and better implementations. Currently, the development is done by an independent Russian team and releases its newer version in every six months.

How OpenCV Works

Computer recognize the image?

- Human eyes provide lots of information based on what they see.
- Machines are facilitated with seeing everything, convert the vision into numbers and store in the memory.
- Computer convert images into numbers : - the pixel value is used to convert images into numbers. A pixel is the smallest unit of a digital image or graphics that can be displayed and represented on a digital display device.



170	283	85	255	221	0
68	138	17	170	119	68
221	0	238	136	0	255
119	255	85	170	136	238
238	17	221	68	119	255
85	170	119	221	17	136

The picture intensity at the particular location is represented by the numbers. In the above image, we have shown the pixel values for a grayscale image consist of only one value, the intensity of the black color at that location.

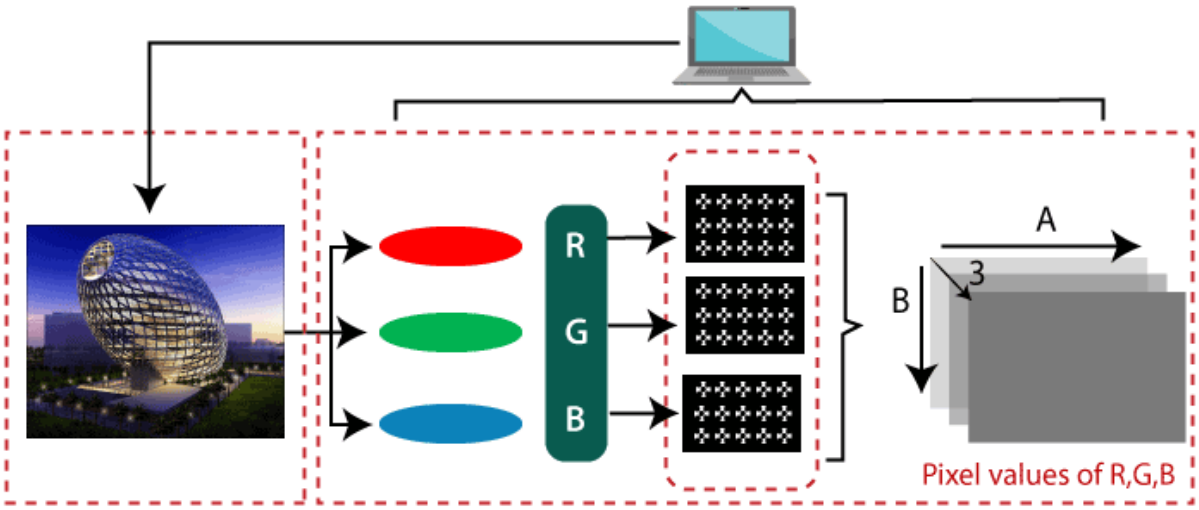
There are two common ways to identify the images:

1. Grayscale

Grayscale images are those images which contain only **two colors black and white**. The contrast measurement of intensity is black treated as the weakest intensity, and white as the strongest intensity. When we use the grayscale image, the computer assigns each pixel value based on its level of darkness.

2. RGB

An RGB is a combination of the red, green, blue color which together makes a new color. The computer retrieves that value from each pixel and puts the results in an array to be interpreted.



OpenCV is used for Computer Vision?

- OpenCV is available for free of cost.
- Since the OpenCV library is written in C/C++, so it is quit fast. Now it can be used with Python.
- It require less RAM to usage, it maybe of 60-70 MB.
- Computer Vision is portable as OpenCV and can run on any device that can run on C.

OpenCV-Python Installation

1. Anaconda

Conda is an open-source package management system and environment management system. It's widely used by data scientists to install and manage software packages and their dependencies.

Spyder, on the other hand, is an open-source integrated development environment (IDE) for Python. It's designed for data scientists and offers powerful features like advanced editing, interactive testing, debugging, and introspection

Linux:

1. Install anaconda:

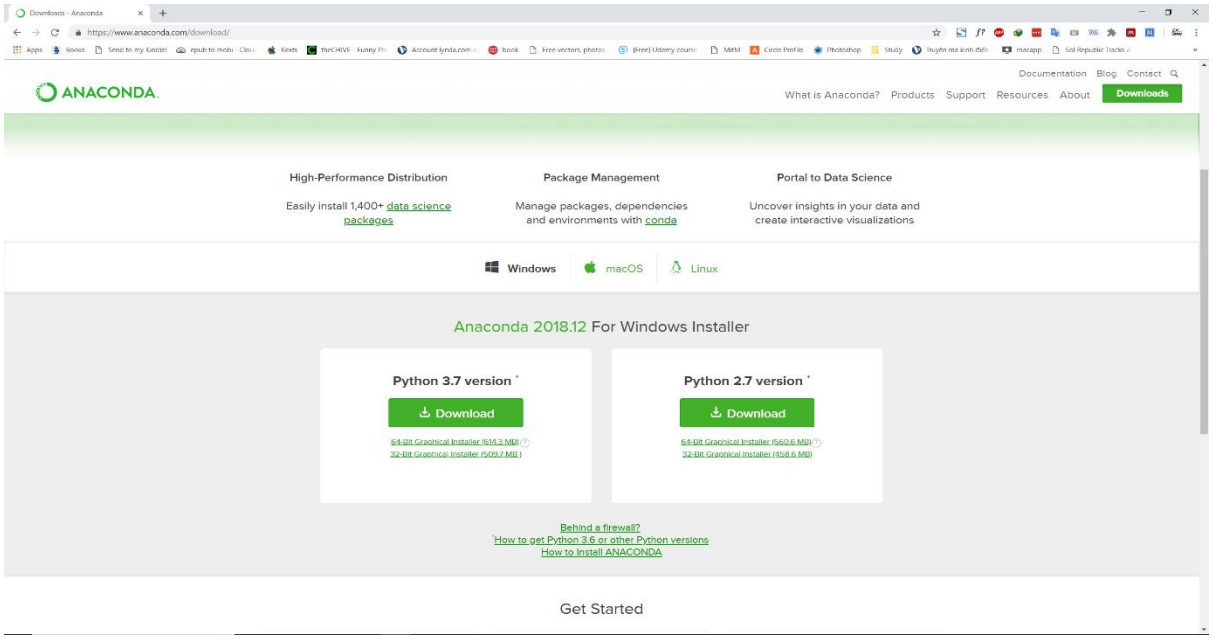
```
wget https://repo.continuum.io/archive/Anaconda3-2018.12-Linux-x86_64.sh
bash Anaconda3-2018.12-Linux-x86_64.sh -b -p $HOME/anaconda
export PATH="$HOME/anaconda/bin:$PATH"
conda update --all
```

2. Install necessary libraries:

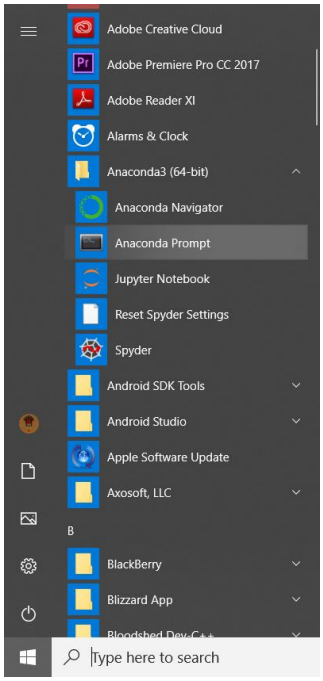
pip install opencv-python==3.4.2.17
pip install opencv-contrib-python==3.4.2.17

Windows:

1. Download and install anaconda environment Python 3.7:
Download: https://www.anaconda.com/download/#windows
Install: http://docs.anaconda.com/anaconda/install/windows/



2. Open Anaconda Prompt
Start Menu / Anaconda3 / Anaconda Prompt

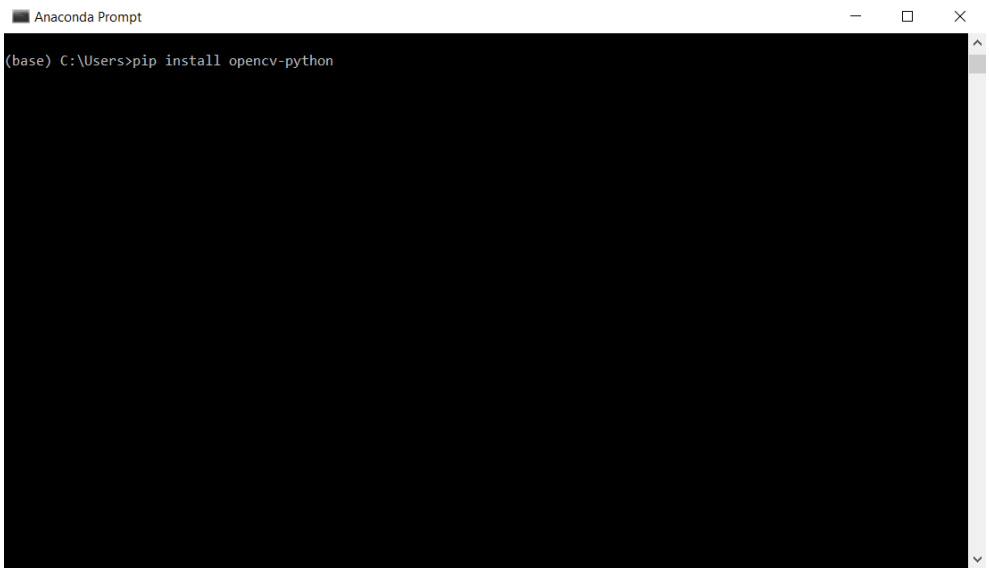


3.In Anaconda Prompt, type commands to install necessary libraries:

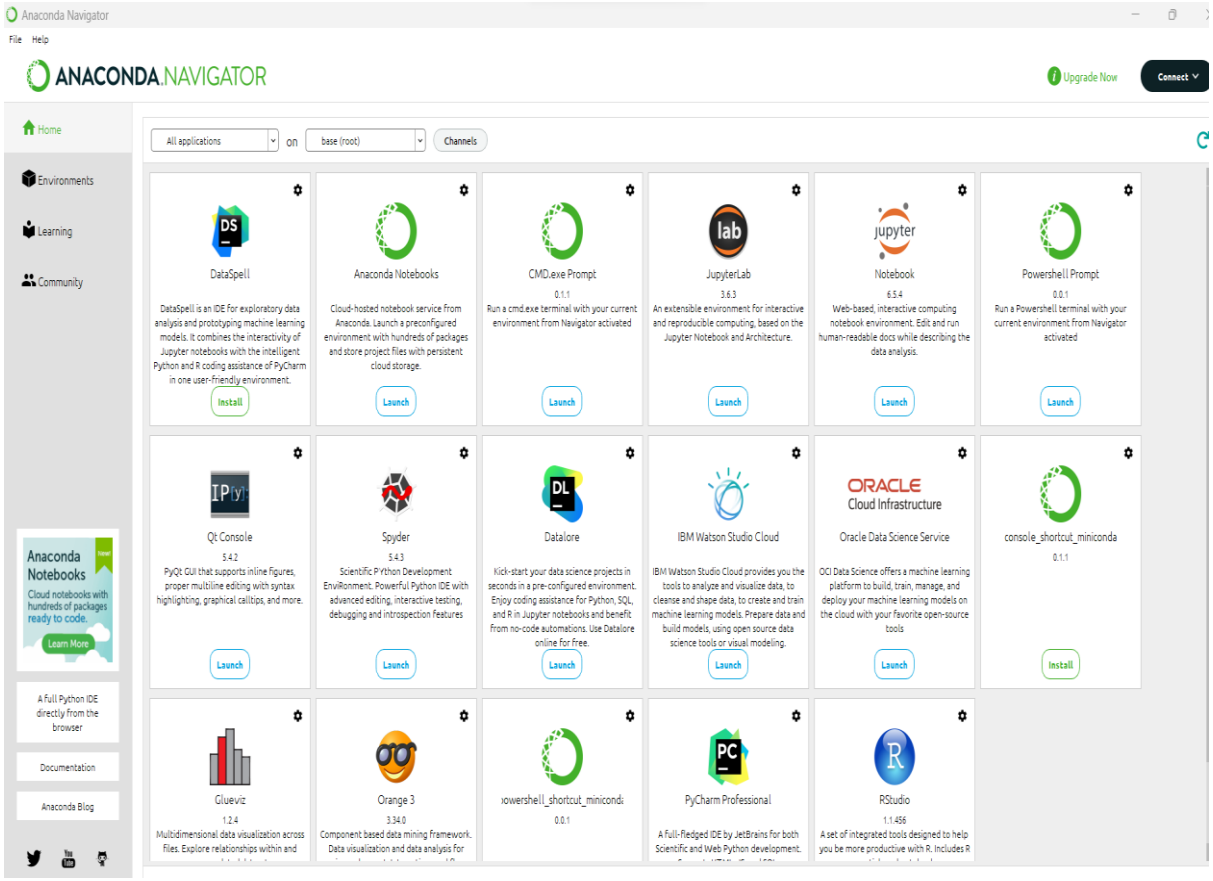
pip install opencv-python==3.4.2.17
pip install opencv-contrib-python==3.4.2.17

Packages for standard desktop environments (Windows, macOS, almost any GNU/Linux distribution)

- Option 1 - Main modules package: pip install opencv-python
- Option 2 - Full package (contains both main modules and contrib/extra modules): pip install opencv-contrib-python

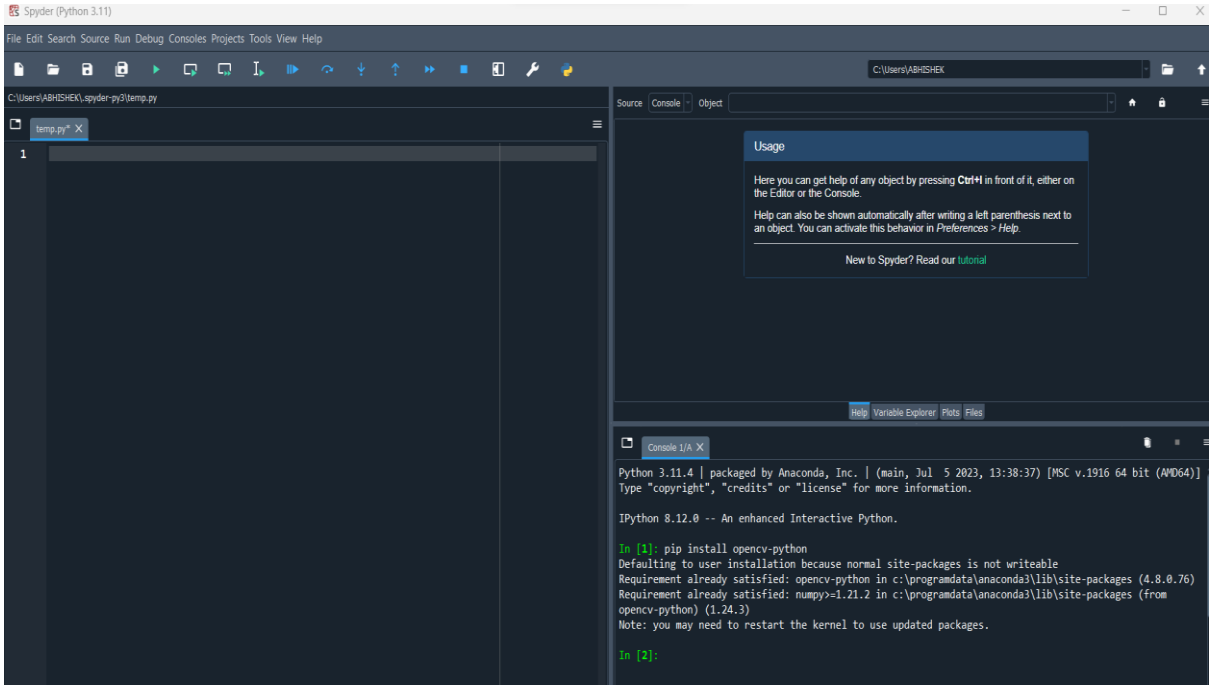


4. Anaconda



5. If OpenCV to be installed on Spyder

Command : pip install opencv-python



1. pip : PIP is a package manager for Python packages

The pip command looks for the package in PyPI, resolves its dependencies, and installs everything in your current Python environment to ensure that requests will work. The pip install <package> command always looks for the latest version of the package and installs it.

Packages required for Open CV

```
In [1]: pip install opencv-python
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: opencv-python in c:\programdata\anaconda3\lib\site-packages (4.8.0.76)
Requirement already satisfied: numpy>=1.21.2 in c:\programdata\anaconda3\lib\site-packages (from
opencv-python) (1.24.3)
Note: you may need to restart the kernel to use updated packages.

In [2]: pip install numpy
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (1.24.3)
Note: you may need to restart the kernel to use updated packages.

In [3]: pip install matplotlib
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: matplotlib in c:\programdata\anaconda3\lib\site-packages (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in c:\programdata\anaconda3\lib\site-packages (from
matplotlib) (1.0.5)
```

1. Numpy :

- NumPy (**Numerical Python**) is an open source Python library that's used in almost every field of science and engineering.
- It's the universal standard for working with numerical data in Python, and it's at the core of the scientific Python and PyData ecosystems.
- The NumPy API is used extensively in Pandas, SciPy, Matplotlib, scikit-learn, scikit-image and most other data science and scientific Python packages.
- **The NumPy library contains multidimensional array and matrix data structures**
- It provides **ndarray**, a homogeneous n-dimensional array object, with methods to efficiently operate on it.
- NumPy can be used to perform a wide variety of mathematical operations on arrays.
- It adds powerful data structures to Python that guarantee efficient calculations with arrays and matrices and it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices.

To access NumPy and its functions import it in your Python code

```
import numpy as np
```

2. Matplotlib :

- Matplotlib is an amazing visualization library in Python for 2D plots of arrays.
- Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack.

3. OpenCV :

- OpenCV-Python is a library of Python bindings designed to solve computer vision problems.

- Import the package:

```
import cv2
```

cv2 is the module import name for opencv-python, "Unofficial pre-built CPU-only OpenCV packages for Python"

PROGRAM 2 : Reading, Writing and Storing Images

Aim : Write a program to read, write and store images

A] Reading an Image

The steps to read and display an image in OpenCV are:

1. Read an image using imread() function.

2. Create a GUI window and display image using imshow() function.

3. Use function waitkey(0) to hold the image window on the screen by the specified number of seconds, o means till the user closes it, it will hold GUI window on the screen.

4. Delete image window from the memory after displaying using destroyAllWindows() function.

1. The imread() function loads image from the specified file and returns it.

Syntax :cv2.imread(filename[,flag])

Parameters:
filename: Name of the file to be loaded
flag: The flag specifies the color type of a loaded image:

- **CV_LOAD_IMAGE_ANYDEPTH -** If we set it as flag, it will return 16-bits/32-bits image when the input has the corresponding depth, otherwise convert it to 8-BIT.
- **CV_LOAD_IMAGE_COLOR -** If we set it as flag, it always return the converted image to the color one.
- **CV_LOAD_IMAGE_GRAYSCALE -** If we set it as flag, it always convert image into the grayscale.

The **imread()** function returns a matrix, if the image cannot be read because of unsupported file format, missing file, unsupported or invalid format. The following file formats are supported.
Window bitmaps - *.bmp, *.dib
JPEG files - *.jpeg, *.jpg, *.jpe
Portable Network Graphics - *.png
Portable image format- *.pbm, *.pgm, *.ppm
TIFF files - *.tiff, *.tif

Return Value:

The cv2.imread() function return a NumPy array if the image is loaded successfully.

2. The imshow() method is used to display an image in a window. The window automatically fits the image size.

Syntax: cv2.imshow(window_name, image)

Parameters:
window_name: A string representing the name of the window in which image to be displayed.
image: It is the image that is to be displayed.
Return Value: It doesn't returns anything.

3. The waitkey() function of Python OpenCV allows users to display a window for given milliseconds or until any key is pressed. It takes time in milliseconds as a parameter and waits for the given time to destroy the window, if 0 is passed in the argument it waits till any key is pressed.

Syntax:waitKey(delay)

delay specifies the time in milliseconds; the currently running thread waits for the keyboard events to happen

waitKey(0) will display the window infinitely until any keypress (it is suitable for image display).
waitKey(1) will display a frame for 1 ms, after which display will be automatically closed.

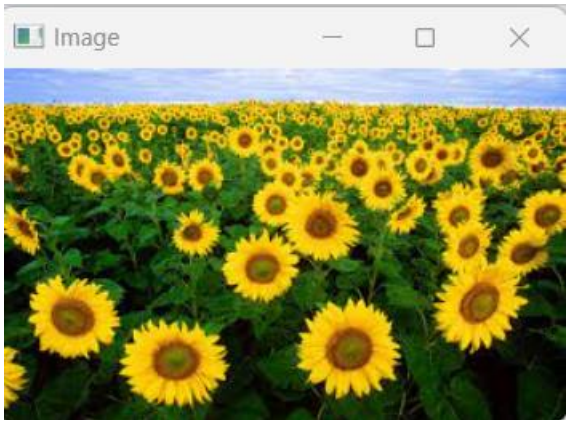
4. The destroyAllWindows() simply destroys all the windows we created. If you want to destroy any specific window, use the function cv2.destroyWindow() where you pass the exact window name as the argument.

Program 1: Read an image

```
#Program to read an image
import numpy as np
import cv2 as cv

# Load the image
image = cv.imread("E:/unnati/DIP/img1.jpg")

# Display the image
# Creating GUI window to display an image on screen
# first Parameter is windows title (should be in string format)
# Second Parameter is image array
cv.imshow("Image", image)

# Wait for the user to press a key
# To hold the window on screen, we use cv2.waitKey methodOnce it detected the
#close input, it will release the control
# To the next line
# First Parameter is for holding screen for specified milliseconds
# It should be positive integer. If 0 pass an parameter, then it will# hold the screen # #
# until user close it.
cv.waitKey(0)

# Close all windows
```

# It is for removing/deleting created GUI window from screenand memory
cv.destroyAllWindows()

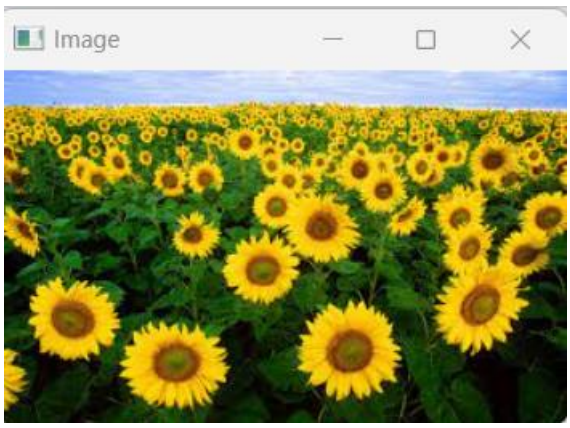
Output :



Program 2: Using waitkey()

```
#Program to read an image
import numpy as np
import cv2 as cv

# Load the image
image = cv.imread("E:/unnati/DIP/img1.jpg")

# Display the image
cv.imshow("Image", image)

# Wait for the user to press a key
cv.waitKey(3000)

# Close all windows
cv.destroyAllWindows()
```

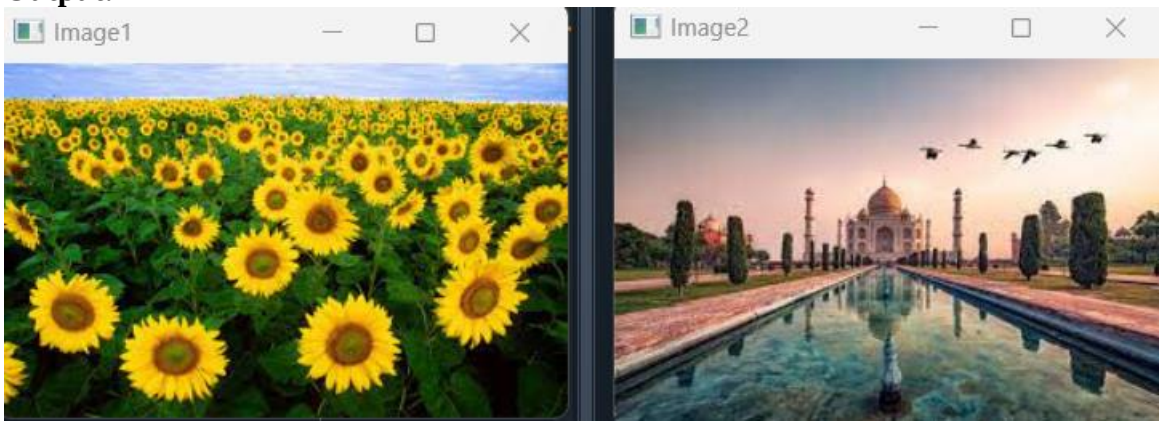
Output : Image will be visible for 3000 ms



Program 3: To read 2 images

```
#Program to read 2 images
import numpy as np
import cv2 as cv

# Load the image
image1 = cv.imread("E:/unnati/DIP/img1.jpg")
image2 = cv.imread("E:/unnati/DIP/img2.jpg")

# Display the image
cv.imshow("Image1", image1)
cv.imshow("Image2", image2)

# Wait for the user to press a key
cv.waitKey(0)
cv.waitKey(0)

# Close all windows
cv.destroyAllWindows()
```

Output:



Program 4: Read an image in different color

```
import numpy as np
import cv2
# Read an image
img_color = cv2.imread("E:/unnati/DIP/img1.jpg",cv2.IMREAD_COLOR)
img_grayscale = cv2.imread("E:/unnati/DIP/img1.jpg",cv2.IMREAD_GRAYSCALE)
img_unchanged =
cv2.imread("E:/unnati/DIP/img1.jpg",cv2.IMREAD_UNCHANGED)

#Displays image inside a window
cv2.imshow('color image',img_color)
cv2.imshow('grayscale image',img_grayscale)
cv2.imshow('unchanged image',img_unchanged)

# Waits for a keystroke
cv2.waitKey(0)

# Destroys all the windows created
```

cv2.destroyAllwindows()

Output:

B] Writing and Storing an Image

The cv2.imwrite() function saves the image with a specified name.

The filename extension and location are also chosen from the first parameter of the function.

This function returns a Boolean value, True will be returned if the image is written successfully. else it will return False.

If the unsupported image format is specified, then the image will be converted to 8-bit unsigned (CV_8U) and saved that way.

Syntax : cv2.imwrite(filename, img[, params])

Parameters:

1. filename: Name/destination where the file has to be saved.
2. img: It takes a ndarray of values to save an image.
3. params: list of Format-specific parameters encoded as pairs.

The image format is automatically decided by OpenCV from the file extension. OpenCV supports *.bmp, *.dib , *.jpeg, *.jpg, *.png,*.webp, *.sr,*.tiff, *.tif etc. image file types.

Program 5: Read an image, then transform it to grey image and save this image data to local file.

```
import numpy as np
import cv2
# read the images
grey_img = cv2.imread("E:/unnati/DIP/img2.jpg")
# read image as grey scale
grey_img = cv2.imread("E:/unnati/DIP/img1.jpg", cv2.IMREAD_GRAYSCALE)
 # save image
status = cv2.imwrite("E:/unnati/DIP/img5.jpg",grey_img)
print("Image written to file-system : ",status)
```

Output:

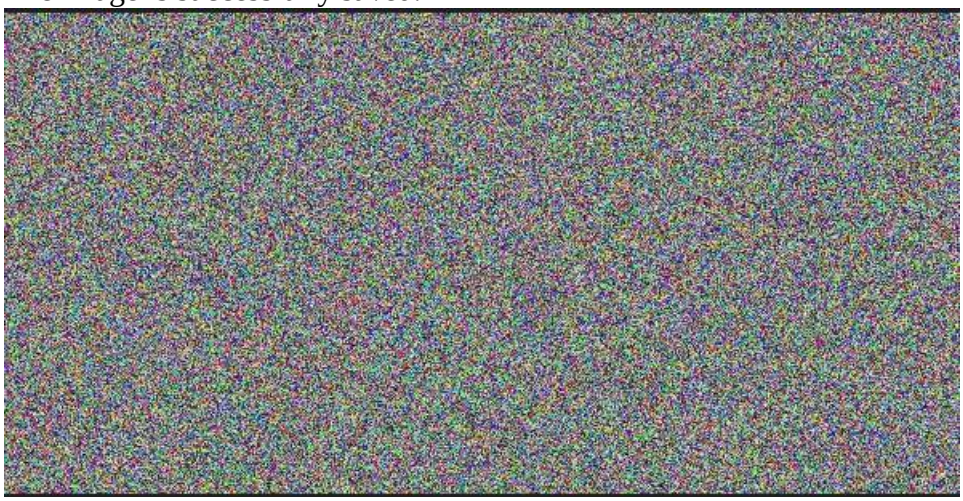
Image written to file-system : True

Program 6: Save image created with random pixel values

```
import numpy as np
import cv2

# read the images
grey_img = np.random.randint(255, size=(300, 600, 3))

isWritten = cv2.imwrite("E:/unnati/DIP/img4.jpg", grey_img)

if isWritten:
print('The image is successfully saved.')
```

Output:

The image is successfully saved.



Program 7: Read an image and save this image data to local file.

```
import numpy as np
import cv2
# read the image
img = cv2.imread("E:/unnati/DIP/img2.jpg")
# read image as grey scale
img = cv2.imread("E:/unnati/DIP/img1.jpg", cv2.IMREAD_COLOR)
 # save image
status = cv2.imwrite("E:/unnati/DIP/img5.jpg",img)
```

Output:

Program 8: program loads an image in grayscale, displays it, save the image if you press 's' and exit, or simply exit without saving if you press *ESC* key.

Python ord() function returns the Unicode code from a given character. This function accepts a string of unit length as an argument and returns the Unicode equivalence of the passed argument

Program :

```
import numpy as np
import cv2

img = cv2.imread("E:/unnati/DIP/img5.jpg",0)
cv2.imshow('image',img)
k = cv2.waitKey(0)
if k == 27:         # wait for ESC key to exit
    cv2.destroyAllWindows()
elif k == ord('s'): # wait for 's' key to save and exit
    cv2.imwrite('messigray.png',img)
    cv2.destroyAllWindows()
```

Output:

Write a program that displays an image then waits from the keyboard interrupt. If the user presses 'c' key, then it will save the color image in our disk, if the user hits 'g' key then it saves the image in grayscale.

Program:

```
import cv2
#Reading images in color and grayscale
color_img = cv2.imread("E:/unnati/DIP/img5.jpg")
```

```
gray_img = cv2.imread("E:/unnati/DIP/img5.jpg",0)

#Displaying the image
cv2.imshow('Sunflower image', color_img)

#Storing the key pressed by user
k = cv2.waitKey(0)

#Check if user hits ‘c’ or ‘g’ key
if( k == ord('c') ):
  cv2.imwrite('E:/unnati/DIP/img8.jpg', color_img )
print("Image is saved color")
  cv2.destroyAllWindows()
if( k == ord('g') ):
  cv2.imwrite('E:/unnati/DIP/img9.jpg', gray_img )
print("Image saved in grayscale")
  cv2.destroyAllWindows()
```

Output :

If c is pressed

Image is saved color

If g is pressed

Image saved in grayscale

PROGRAM 3: Read an image as grayscale

Aim : Write a program in python to read an image in grayscale

Grayscaling is the process of converting an image from other color spaces e.g. RGB, CMYK, HSV, etc. to shades of gray. It varies between complete black and complete white.

Importance of grayscaling

- **Dimension reduction:** For example, In RGB images there are three color channels and three dimensions while grayscale images are single-dimensional.
- **Reduces model complexity:** Consider training neural articles on RGB images of 10x10x3 pixels. The input layer will have 300 input nodes. On the other hand, the same neural network will need only 100 input nodes for grayscale images.
- **For other algorithms to work:** Many algorithms are customized to work only on grayscale images e.g. Canny edge detection function pre-implemented in the OpenCV library works on Grayscale images only.

Three methods

- Convert image to grayscale with imread() function
- Convert image to grayscale using cvtColor() function
- Using the pixel manipulation (Average method)

Method 1: Using imread() function
imread() function is used to read an image in OpenCV but there is one more parameter to be considerd, that is flag which decides the way image is read. There three flag defined in OpenCV..

1. cv2.IMREAD_COLOR or 1
2. cv2.IMREAD_GRAYSCALE or 0
3. cv2.IMREAD_UNCHANGED or -1

So to convert the color image to grayscale we will be using cv2.imread("image-name.png",0) or you can also write cv2.IMREAD_GRAYSCALE in the place of 0 as it also denotes the same constant.

Program:

```
import cv2

# Reading color image as grayscale
gray = cv2.imread("E:/unnati/DIP/img2.jpg",0)

# Showing grayscale image
cv2.imshow("Grayscale Image", gray)

# waiting for key event
cv2.waitKey(0)
```

```
# destroying all windows
cv2.destroyAllWindows()
```

INPUT :

OUTPUT:

Method 2: Using cvtColor() function
cvtColor() function in OpenCV is very helpful in converting color channels from one to another such as BRG to HSV or BRG to RGB. The same method can be used to convert BRG to GRAY by using the cv2.cvtColor(img,cv2.BGR2GRAY)

BRG (Blue, red, green)
HSV (Hue, Saturation, value)

cv2.cvtColor() function to convert a given color image to grayscale

Syntax of cvtColor() : cv2.cvtColor(src, code, dst, dstCn)

where

Parameter	Description
src	Input/source image array.
code	Color space conversion code
dst	[Optional] Output array of the same size and type as src.
dstCn	[Optional] Number of channels in the destination image.

Steps to convert an image to grayscale

1. Read an image into an array using cv2.imread() function.
2. Call the cv2.cvtColor() function and pass the input image array and color code of cv2.COLOR_BGR2GRAY as arguments. It returns the grayscale image array.
3. Save the grayscale image using cv2.imwrite() function.

Program:

```
import cv2

# Reading color image
img = cv2.imread("E:/unnati/DIP/img2.jpg")

# Converting color image to grayscale image
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Showing the converted image
cv2.imshow("Converted Image",gray)

# waiting for key event
cv2.waitKey(0)

# destroying all windows
cv2.destroyAllWindows()
```

INPUT:

OUTPUT

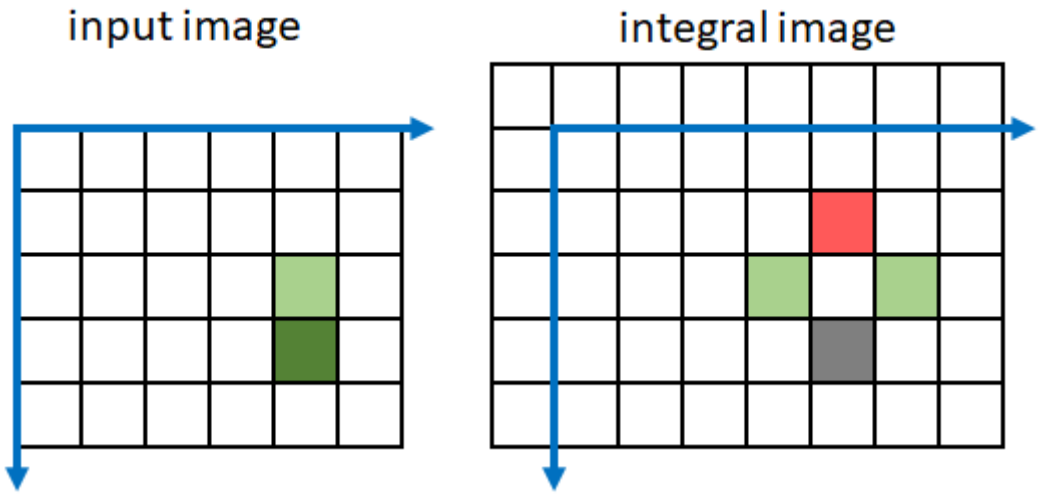
Method 3: Using the pixel manipulation (Average method)

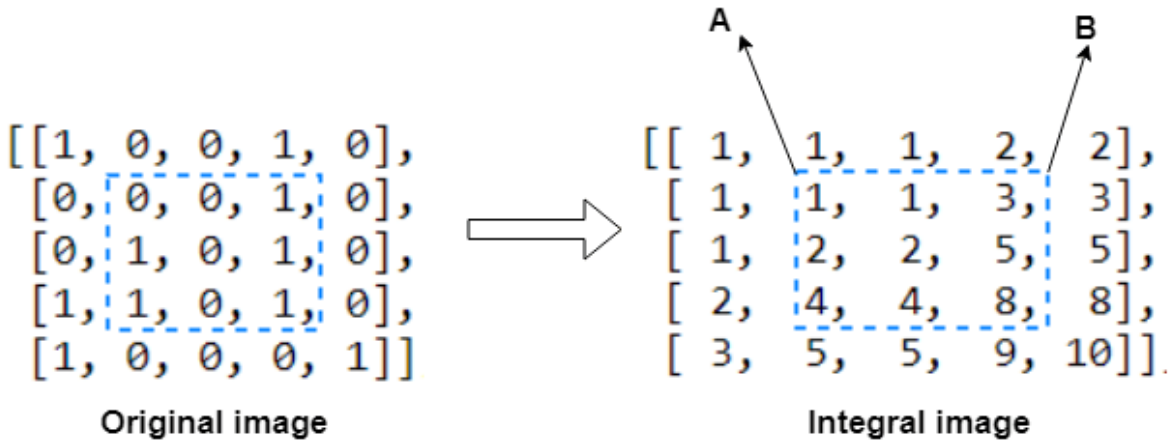
img.shape returns (Height, Width, Number of Channels)
where

1. Height represents the number of pixel rows in the image or the number of pixels in each column of the image array.
2. Width represents the number of pixel columns in the image or the number of pixels in each row of the image array.
3. Number of Channels represents the number of components used to represent each pixel.

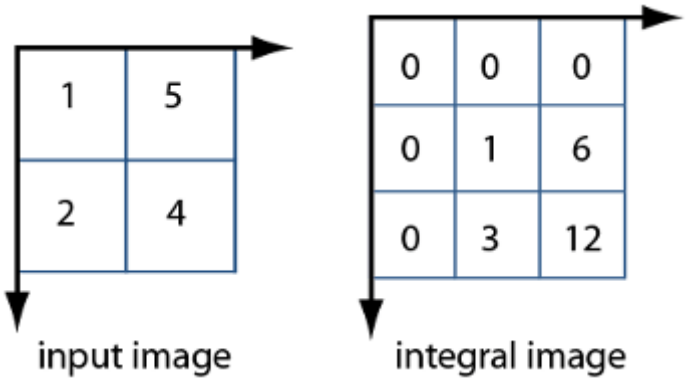
eg: Number of Channels = 4 represent Alpha, Red, Green and Blue channels.

RGB images are converted to grayscale using the formula **gray=(red+green+blue)/3 or gray=0.299red+0.587green+0.114blue**





Sum = 8 + 1 - 2 - 2 = 5



img[i, j] = sum(img[i, j]) * 0.33

Program :

```
# Import opencv
import cv2

# Load the input image
img = cv2.imread("E:/unnati/DIP/img2.jpg")
# get dimensions of image
dimensions = img.shape

# height, width, number of channels in image
height = img.shape[0]
width = img.shape[1]
channels = img.shape[2]
```

```
print('Image Dimension    : ',dimensions)
print('Image Height       : ',height)
print('Image Width        : ',width)
print('Number of Channels : ',channels)

# Obtain the dimensions of the image array
# using the shape method
(row, col) = img.shape[0:2]

# Take the average of pixel values of the BGR Channels
# to convert the colored image to grayscale image
for i in range(row):
    for j in range(col):
        # Find the average of the BGR pixel values
img[i, j] = sum(img[i, j]) * 0.33

cv2.imshow('Grayscale Image', img)
cv2.waitKey(0)

# Window shown waits for any key pressing event
cv2.destroyAllWindows()
```

INPUT :

OUTPUT:

```
Image Dimension    :  (184, 274, 3)
Image Height       :  184
Image Width        :  274
Number of Channels :  3
```


Grayscale Image

PROGRAM 4: Reading Images as RGB

Aim : Write a program in python to read an image as RGB

Each digital image is made up of a collection of pixels. Each pixel in an image has a color, which may be represented as an RGB value/triplet/tuple. PIL is the Python Imaging Library which provides the python interpreter with image editing capabilities.

- Python Imaging Library (expansion of PIL) is the image processing package for Python language.
- It incorporates lightweight image processing tools that aids in editing, creating and saving images.
- Pillow supports a large number of image file formats including BMP, PNG, JPEG, and TIFF. The library encourages adding support for newer formats in the library by creating new file decoders.
- This module is not preloaded with Python. So to install it execute the following command in the command-line:

pip install pillow

Commands in Pillow

1. Opening an image using open(): The PIL.Image.Image class represents the image object. This class provides the open() method that is used to open the image.

   img = Image.open(r"test.png")

2. Displaying the image using show(): This method is used to display the image. For displaying the image Pillow first converts the image to a .png format (on Windows OS) and stores it in a temporary buffer and then displays it. Therefore, due to the conversion of the image format to .png some properties of the original image file format might be lost (like animation). Therefore, it is advised to use this method only for test purposes.

   img.show()

   Program 1: Program to read RGB pixel value at any location

```
import numpy as np
from PIL import Image
# Read in image file
img = Image.open("E:/unnati/DIP/img3.jpg")
#Create a PIL.Image object

red_image_rgb = img.convert("RGB")
```

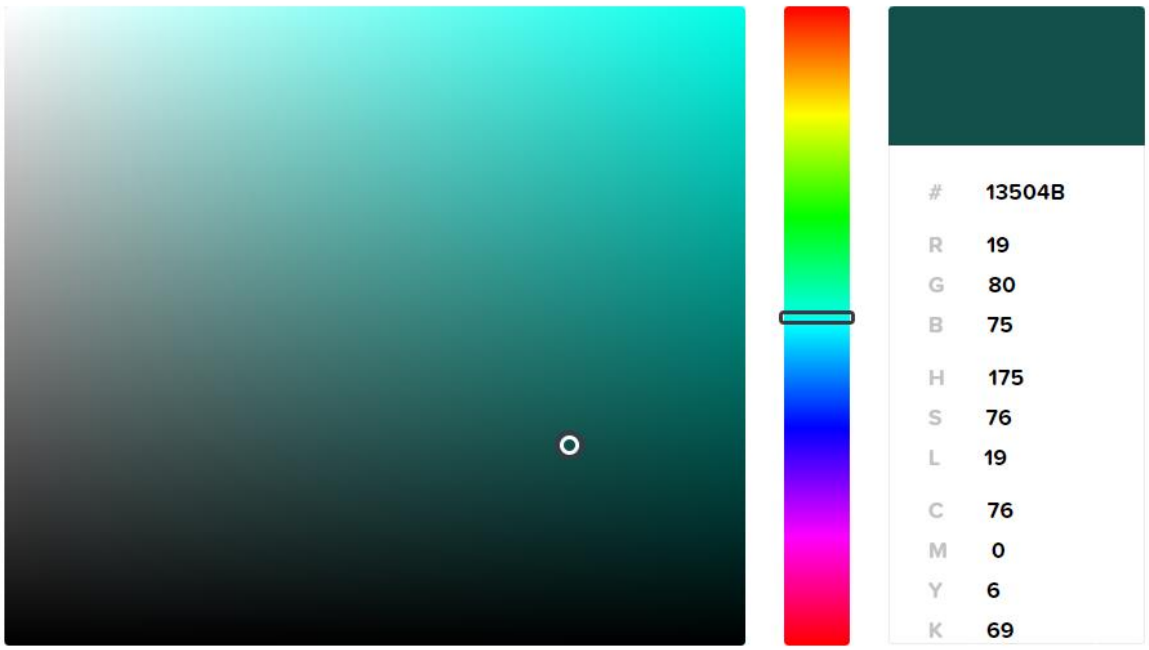
#Convert to RGB colorspace

rgb_pixel_value = red_image_rgb.getpixel((10,15))
#Get color from (x, y) coordinates

print(rgb_pixel_value)

Input:

Output:
(19, 80, 75)



Program 2: Program to read RGB pixel value of whole image

import numpy as np
from PIL import Image
# Read in image file

```
img = Image.open("E:/unnati/DIP/img3.jpg")
#Create a PIL.Image object

image_rgb = img.convert("RGB")
#Convert to RGB colorspace

#Get Width and Height of Image

width, height = image_rgb.size
#Iterate through all pixels of Image and get R, G, B value from that pixel

for x in range(0, width):
    for y in range(0, height):
        r, g, b = image_rgb.getpixel((x,y))
        print(image_rgb.getpixel((x,y)))
```

Input:

Output:

```
(200, 226, 223)
(205, 225, 224)
(205, 225, 224)
(206, 226, 225)
(206, 226, 225)
(207, 227, 226)
……..
(168, 204, 142)
(170, 206, 144)
(171, 207, 145)
(173, 209, 147)
(174, 210, 148)
(175, 211, 149)
(178, 214, 152)
(178, 214, 152)
(178, 214, 152)
```

Program 3: Program to calculate total number of red, green and blue color in an image

```
import numpy as np
from PIL import Image
# Read in image file
img = Image.open("E:/unnati/DIP/img3.jpg")
#Create a PIL.Image object

image_rgb = img.convert("RGB")
#Convert to RGB colorspace

#Get Width and Height of Image

width, height = image_rgb.size
#Iterate through all pixels of Image and get R, G, B value from that pixel

for x in range(0, width):
    for y in range(0, height):
        r, g, b = image_rgb.getpixel((x,y))


r_total = 0
g_total = 0
b_total = 0

for x in range(0, width):
    for y in range(0, height):
        r, g, b = img.getpixel((x,y))
r_total += r
g_total += g
b_total += b
print(r_total, g_total, b_total)
```

Input:

Output:

7496330 9370231 8011306

PROGRAM 5: Image Conversion :Colored Image to Gray Scale

Aim : Write a program in python to convert a colored image to gray scale

Color Spaces

Color spaces are a way to represent the color channels present in the image that gives the image that particular hue.
There are several different color spaces and each has its own significance. Some of the popular color spaces

1. RGB (Red, Green, Blue)
2. CMYK (Cyan, Magenta, Yellow, Black)
3. HSV (Hue, Saturation, Value)
4. BGR (Blue, Green, Red)

BGR color space: OpenCV's default color space is RGB. However, it actually stores color in the BGR format.
It is an additive color model where the different intensities of Blue, Green and Red give different shades of color.

HSV color space: It stores color information in a cylindrical representation of RGB color points. It attempts to depict the colors as perceived by the human eye. Hue value varies from 0-179, Saturation value varies from 0-255 and Value value varies from 0-255. It is mostly used for color segmentation purpose.

CMYK color space: Unlike, RGB it is a subtractive color space. The CMYK model works by partially or entirely masking colors on a lighter, usually white, background.

Program 1: Program to display monochrome version of the image

```
import cv2
image = cv2.imread("E:/unnati/DIP/img1.jpg")
B, G, R = cv2.split(image)
# Corresponding channels are separated
cv2.imshow("RGB Circles ", image)
cv2.waitKey(0)

cv2.imshow("blue", B)
cv2.waitKey(0)

cv2.imshow("Green", G)
cv2.waitKey(0)
```

```
cv2.imshow("red", R)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Observation :Opens the img1.jpg image in a window titled “RGB Circles”. 3 other windows are opened in windows titled Red, Green and Blue displaying monochrome version of rgbcircle with the corresponding color shown in white and all other colors shown a black.

Input :

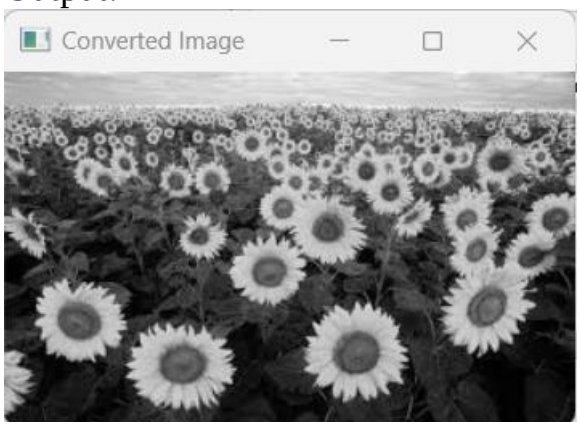
Output:

Program 2: Program to convert colortograyscale

```
#COLOR TO GRAYSCALE

import cv2
# Reading color image

img = cv2.imread("E:/unnati/DIP/img1.jpg")
# Converting color image to grayscale image
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Showing the converted image
cv2.imshow("Converted Image",gray)
# waiting for key event
cv2.waitKey(0)
# destroying all windows
cv2.destroyAllWindows()
```

Input:

Output:

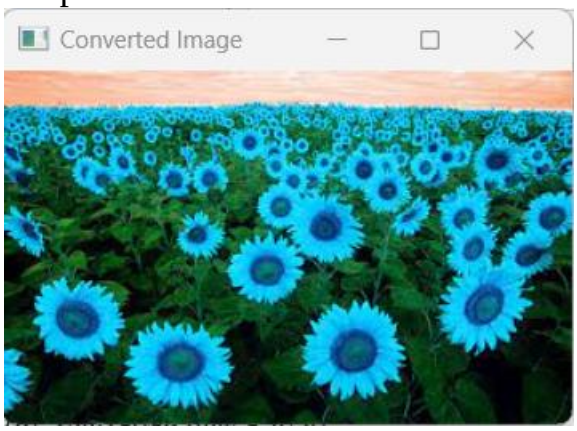


Program 3: Program to convert RGB to BGR

```
#RGB to BGR
import cv2
# Reading color image
img = cv2.imread("E:/unnati/DIP/img1.jpg")
# Converting color image to grayscale image
rgb = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
# Showing the converted image
cv2.imshow("Converted Image",rgb)
# waiting for key event
cv2.waitKey(0)
# destroying all windows
cv2.destroyAllWindows()
```

Input:

Output:



Program 3: Program to convert BGR to RGB

```
#BGR to RGB
import cv2
# Reading color image
img = cv2.imread("E:/unnati/DIP/img1.jpg")
# Converting color image to grayscale image
rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# Showing the converted image
cv2.imshow("Converted Image",rgb)
# waiting for key event
cv2.waitKey(0)
# destroying all windows
cv2.destroyAllWindows()
```

Input:

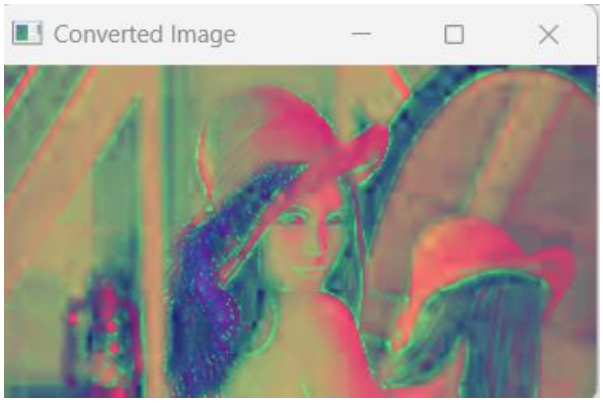
Output:

Program 4: Program to convert BGR to HSV

```
#BGR to HSV
import cv2
# Reading color image
img = cv2.imread("E:/unnati/DIP/img1.jpg")
# Converting color image to grayscale image
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
# Showing the converted image
cv2.imshow("Converted Image",hsv)
# waiting for key event
cv2.waitKey(0)
# destroying all windows
cv2.destroyAllWindows()
```

Input :

Output:



VIEWING IMAGES USING MATPLOT

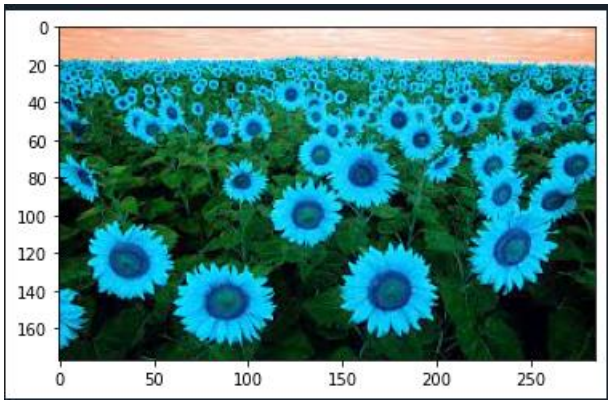
Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

PROGRAM 5

```
import cv2
import matplotlib.pyplot as plt

# Reading color image
img=cv2.imread("E:/unnati/DIP/img1.jpg")

#Displaying image in plot
plt.imshow(img)
```

INPUT

OUTPUT:


0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

PROGRAM 6: Image Conversion :Colored Image to Binary

Aim : Write a program in python to convert a colored image to binary

Binary images are images for which the pixels have only two possible intensity values. These are displayed as black and white, where each pixel of the image has the value 0 or 255 representing black and white.

Importance of OpenCV Binary conversion

1. Binary images are particularly useful for image processing as they allow easy separation of an object from the background. The process of segmentation allows the labeling of each pixel as the background pixel or the object pixel and assigns corresponding intensity values.

2. Binary images find several applications as they are the simplest images to process, they are an impoverished representation of the image information.

3. These images are used when necessary information is to be obtained by the silhouette of the object and the silhouette of the object can be obtained easily.

4. Binary images find use in digital image processing as masks or thresholding.

Thresholding in OpenCV

Thresholding is an image segmentation process, where a common function is applied to the pixels of an image to make images easier to analyze. The same threshold value is used for each pixel value. If the pixel value is less than the threshold value, it is updated to 0, otherwise, it is updated to a maximum value. In OpenCV, cv2.threshold() is used for thresholding. In thresholding, an image is converted from color or grayscale into a binary image.

Syntax:
cv2.threshold(source, thresholdValue, maxVal, thresholdingTechnique)

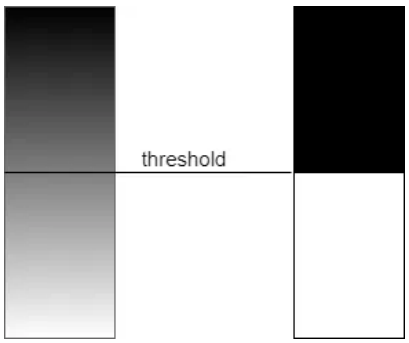
Parameters:

- source: Input Image array (must be in Grayscale).
- thresholdValue: Value of Threshold below and above which pixel values will change accordingly.
- maxVal: Maximum value that can be assigned to a pixel.
- thresholdingTechnique: The type of thresholding to be applied.

The different Simple Thresholding Techniques are:

1. cv2.THRESH_BINARY: If pixel intensity is greater than the set threshold, value set to 255, else set to 0 (black).
2. cv2.THRESH_BINARY_INV: Inverted or Opposite case of cv2.THRESH_BINARY.
3. cv.THRESH_TRUNC: If pixel intensity value is greater than threshold, it is truncated to the threshold. The pixel values are set to be the same as the threshold. All other values remain the same.
4. cv.THRESH_TOZERO: Pixel intensity is set to 0, for all the pixels intensity, less than the threshold value.
5. cv.THRESH_TOZERO_INV: Inverted or Opposite case of cv2.THRESH_TOZERO.

$$dst(x,y) = \begin{cases} maxval & \text{if } src(x,y) > thresh \\ 0 & \text{otherwise} \end{cases}$$



Convert Image to Binary or Black and White

PROGRAM 01

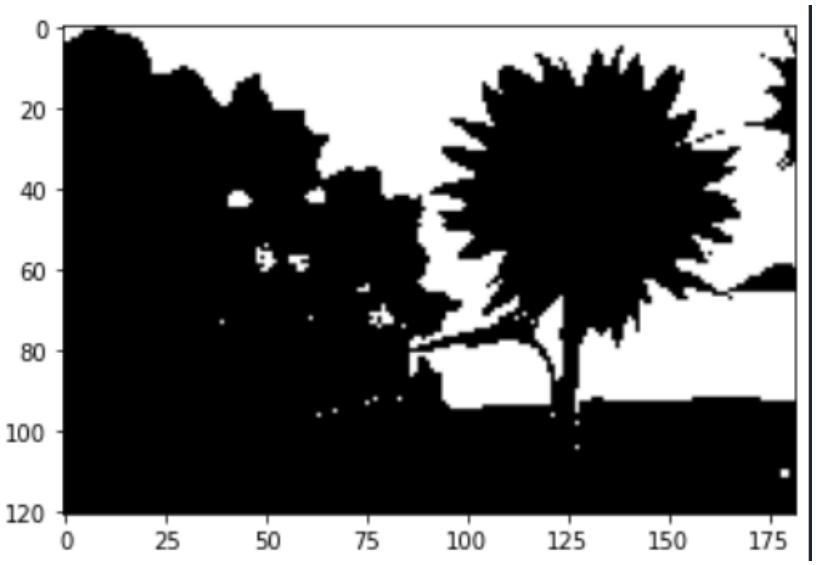
Aim: Program to convert a coloredimage to binary image

```
# Importing opencv
import cv2
# Importing matplotlib.pyplot
import matplotlib.pyplot as plt
# Reading the image
image = cv2.imread("E:/unnati/DIP/img10.jpeg")
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
# Displaying the original image
plt.imshow(image)
# Converting into grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Displaying the converted image
```

```
plt.imshow(gray_image, cmap='gray')
# Converting to binary image using thresholding
(thresh, binary_image) = cv2.threshold(gray_image, 175, 255,
cv2.THRESH_BINARY)
# Displaying binary image
plt.imshow(binary_image, cmap='gray')
```

INPUT :

OUTPUT:



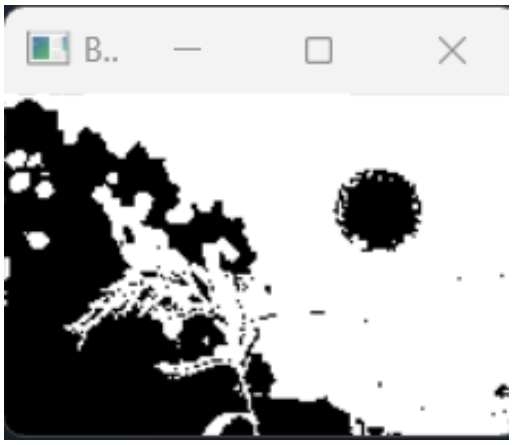
PROGRAM 02
Aim: Program to convert a coloredimage to binary image

```
# import required libraries
import cv2
# load the input image
img = cv2.imread("E:/unnati/DIP/img10.jpeg")
# convert the input image to grayscale
```

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# apply thresholding to convert grayscale to binary image
ret,thresh = cv2.threshold(gray,70,255, cv2.THRESH_BINARY)
# Display the Binary Image
cv2.imshow("Binary Image", thresh)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT :

OUTPUT:



PROGRAM 03

Aim: Program to convert a coloredimage to binary image(using matplotlib)

```
# import required libraries
import cv2
import matplotlib.pyplot as plt
# load the input image
img = cv2.imread("E:/unnati/DIP/img10.jpeg")
# convert BGR to RGB to display using matplotlib
imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# convert the input image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# apply thresholding to convert grayscale to binary image
```

```
ret,thresh = cv2.threshold(gray,70,255,0)
# display Original, Grayscale and Binary Images
plt.subplot(131),plt.imshow(imgRGB,cmap = 'gray'),plt.title('Original Image'),
plt.axis('off')
plt.subplot(132),plt.imshow(gray,cmap = 'gray'),plt.title('Grayscale
Image'),plt.axis('off')
plt.subplot(133),plt.imshow(thresh,cmap = 'gray'),plt.title('Binary
Image'),plt.axis('off')
plt.show()
```

INPUT :

OUTPUT:

PROGRAM 04

Aim: Program to convert a coloredimage to binary image by applying different thresholding techniques

```
import cv2
import numpy as np
# load the input image
image1 = cv2.imread("E:/unnati/DIP/img10.jpeg")
# to convert the image in grayscale
img = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
# applying different thresholding with threshold value = 120
ret, thresh1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY)
```

```
ret, thresh2 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
ret, thresh3 = cv2.threshold(img, 120, 255, cv2.THRESH_TRUNC)
ret, thresh4 = cv2.threshold(img, 120, 255, cv2.THRESH_TOZERO)
ret, thresh5 = cv2.threshold(img, 120, 255, cv2.THRESH_TOZERO_INV)
# the window showing output images
cv2.imshow('Binary Threshold', thresh1)
cv2.imshow('Binary Threshold Inverted', thresh2)
cv2.imshow('Truncated Threshold', thresh3)
cv2.imshow('Set to 0', thresh4)
cv2.imshow('Set to 0 Inverted', thresh5)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

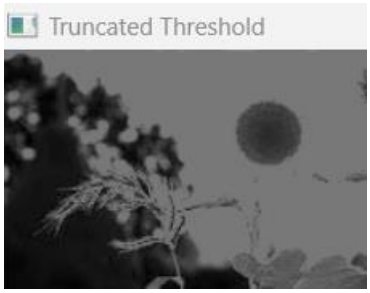
INPUT :

OUTPUT:

1.BINARY THRESHOLD

2.BINARY THRESHOLD INVERTED

1. TRUNCATED THRESHOLD



2. SET TO 0

3. SET TO 0 INVERTED

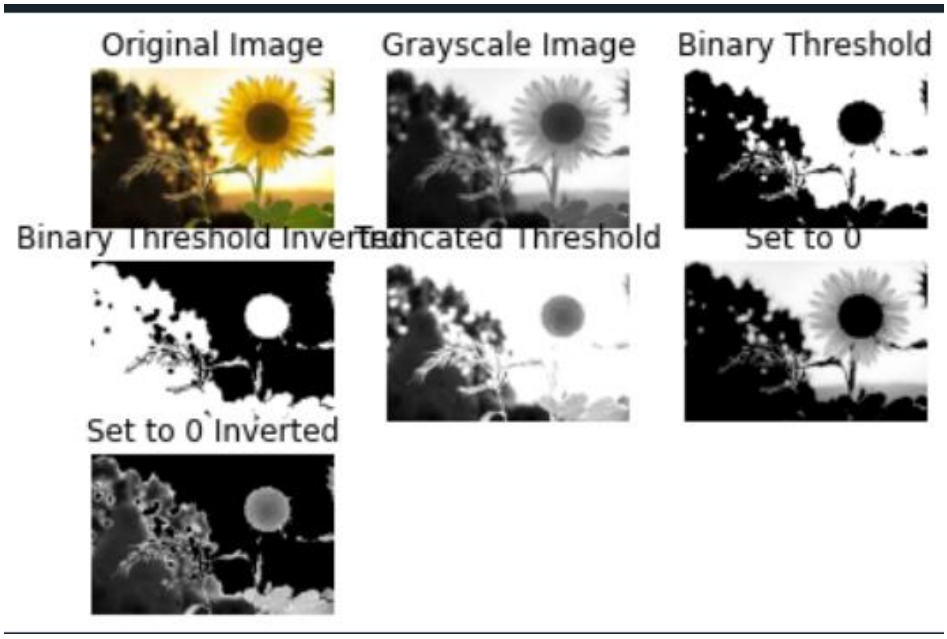
PROGRAM 04
Aim: Program to convert a coloredimage to binary image by applying different thresholding techniques using matplotlib

```
# organizing imports
import cv2
import matplotlib.pyplot as plt
# load the input image
img = cv2.imread("E:/unnati/DIP/img10.jpeg")
# convert BGR to RGB to display using matplotlib
imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# to convert the image in grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# applying different thresholding with threshold value = 120
```

```
ret, thresh1 = cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY)
ret, thresh2 = cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY_INV)
ret, thresh3 = cv2.threshold(gray, 120, 255, cv2.THRESH_TRUNC)
ret, thresh4 = cv2.threshold(gray, 120, 255, cv2.THRESH_TOZERO)
ret, thresh5 = cv2.threshold(gray, 120, 255, cv2.THRESH_TOZERO_INV)
# display Original, Grayscale and Binary Images
plt.subplot(331),plt.imshow(imgRGB,cmap = 'gray'),plt.title('Original Image'),
plt.axis('off')
plt.subplot(332),plt.imshow(gray,cmap = 'gray'),plt.title('Grayscale
Image'),plt.axis('off')
plt.subplot(333),plt.imshow(thresh1,cmap = 'gray'),plt.title('Binary
Threshold'),plt.axis('off')
plt.subplot(334),plt.imshow(thresh2,cmap = 'gray'),plt.title('Binary Threshold
Inverted'),plt.axis('off')
plt.subplot(335),plt.imshow(thresh3,cmap = 'gray'),plt.title('Truncated
Threshold'),plt.axis('off')
plt.subplot(336),plt.imshow(thresh4,cmap = 'gray'),plt.title('Set to
0'),plt.axis('off')
plt.subplot(337),plt.imshow(thresh5,cmap = 'gray'),plt.title('Set to 0
Inverted'),plt.axis('off')
plt.show()
```

INPUT :

OUTPUT:


Original Image
Grayscale Image
Binary Threshold
Binary Threshold Inverted
Truncated Threshold
Set to 0
Set to 0 Inverted

PROGRAM 7: Processing – Blur- Averaging, Gaussian

Aim : Write a program in python to blur an image using Averaging and Gaussian filter

Theory:

The blurring or smoothening techniques applied on an image remove outlier pixels that may constitute noise in an image. Smoothing and blurring operations are important steps in computer vision and image processing.

Blurring is the process of applying a low-pass filter to an image. In the field of computer vision, a low pass filter refers to the removal of noise from an image while not disrupting the majority of its regions.
OpenCV blurs an image by applying a Kernel. A kernel describes the change in the value of pixels depending on the combination of neighboring pixels selected. The kernel is applied to each pixel of the image to give the final result and this process is termed convolution.

Why is blurring important in image processing?
1. Removal of noise: A high pass signal is considered as noise in an image and by application of a low pass filter to an image, the noise is restricted.

2. Removal of high-frequency content: It removes high-frequency content which might not be useful for us such as noise and edges. It reduces the details of an image and aids in the application of other algorithms to the image. By reducing the details, we can recognize other features more easily.

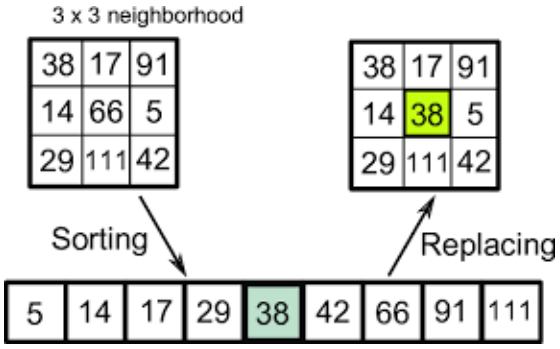
3. Removal of low-intensity edges: The value of image intensity change is not significant from one side of the abruption encountered to another and hence it is discarded.

Types of filters
Blurring can be achieved by many ways. The common type of filters that are used to perform blurring are.

- Mean filter
- Weighted average filter
- Gaussian filter

Median filter

The median filter is a simple sliding-window to replace each pixel value in an image with the median value of its neighbors, including itself.



Averaging / Mean Filter

Averaging is the blurring technique where the image is normalized. It replaces the central elements with the calculated average of pixel values under the kernel area.

This is done by convolving the image with a normalized box filter.

It simply takes the average of all the pixels under kernel area and replaces the central element with this average. This is done by the function cv2.blur() or cv2.boxFilter().

A 3x3 normalized box filter would look like this:

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

OpenCV provides the cv2.blur() function for averaging applications. The kernel height and width need to be defined in this function.

Syntax

cv2.blur(src, dst, ksize, anchor, borderType)

Parameters

src: Source image or input image

dst: Output image

ksize: Kernel size

anchor: Anchor points

borderType: The type of border

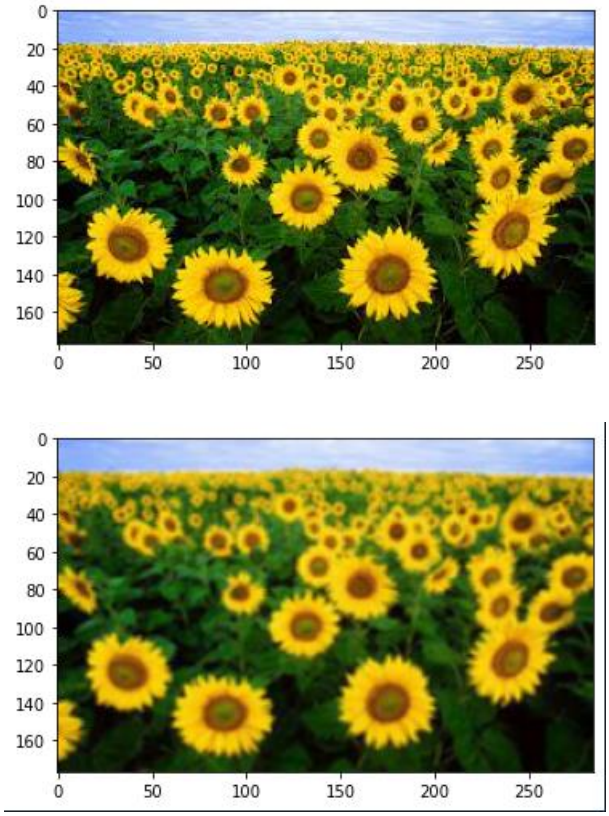
Program 1: Implement Averaging/ Mean filter

```
# Importing OpenCV
import cv2
# Importing numpy
import numpy as np
# Importing matplotlib.pyplot
import matplotlib.pyplot as plt
# Reading the image
img = cv2.imread("E:/unnati/DIP/img1.jpg")
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# Displaying the original image
```

```
plt.imshow(img)
# Averaging the image
img = cv2.blur(img, (3,3))
# Displaying the blurred image
plt.imshow(img)
```

INPUT :

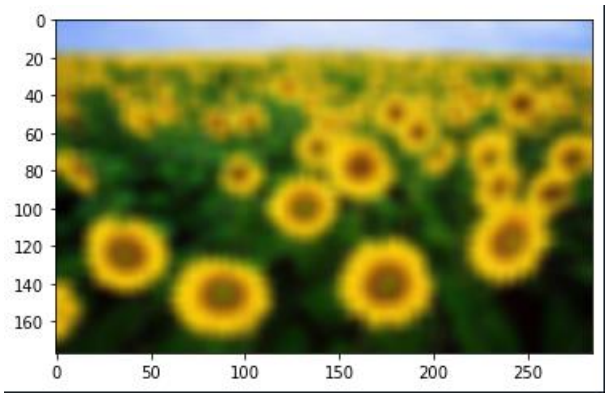
OUTPUT



If Input is

```
img = cv2.blur(img, (9,9))
# Displaying the blurred image
plt.imshow(img)
```

Output :



Gaussian

This technique uses a Gaussian distribution to construct the kernel.Gaussian blurring applies a weighted average, where pixels that are closer to the center of the kernel have more weight than pixels that are further away.

In the Gaussian blurring method, a Gaussian kernel is used.

It is done with the function, cv2.GaussianBlur().
Gaussian blur replaces the central elements with the calculated weighted mean of pixel values under the kernel area.
In Gaussian blurring, pixels closer to the central element, contribute more to the weight.
Gaussian blurring is used to remove noise following a gaussian distribution.

The kernel height and width need to be defined for this function which should be positive and odd. The standard deviation in X and Y directions also needs to be specified. If the standard deviation in one direction is specified, then the other is taken as the same. If both deviations are zero, then the standard deviation is calculated using the kernel size.

It is a widely used effect in graphics software, typically to reduce image noise and reduce detail. It is also used as a preprocessing stage before applying our machine learning or deep learning models.

$$1/16 \quad \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

E.g. of a Gaussian kernel(3×3)

Syntax
cv2.GaussianBlur(src, ksize, dst, sigmaX,sigmaY, borderType)

Parameters
src: Source image or input image
dst: Output image

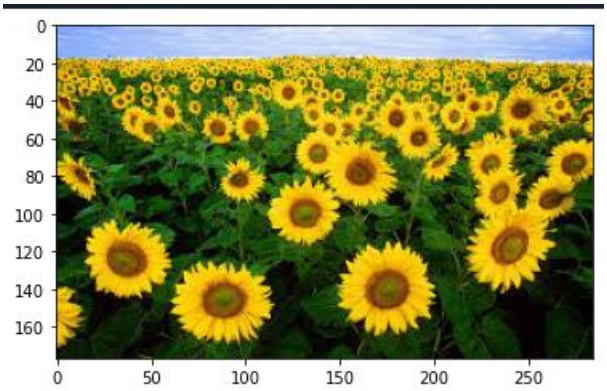
ksize: Kernel size. Height and width must be positive and odd.
sigmaX: Standard deviation in the X direction
sigmaY: Standard deviation in the Y direction
borderType: The type of border
Possible values are: cv2.BORDER_CONSTANT cv2.BORDER_REPLICATE cv2.BORDER_REFLECT cv2.BORDER_WRAP cv2.BORDER_REFLECT_101 cv2.BORDER_TRANSPARENT cv2.BORDER_REFLECT101 cv2.BORDER_DEFAULT cv2.BORDER_ISOLATED

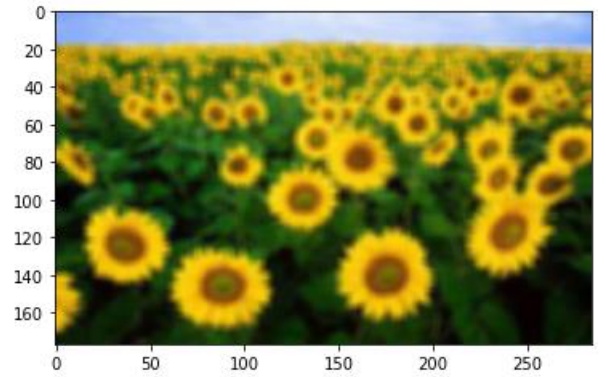
Program 2: Implement Gaussian filter

```
# Importing OpenCV
import cv2
# Importing numpy
import numpy as np
# Importing matplotlib.pyplot
import matplotlib.pyplot as plt
# Reading the image
img = cv2.imread("E:/unnati/DIP/img1.jpg")
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# Displaying the original image
plt.imshow(img)
# Averaging the image
img = cv2.GaussianBlur(img, (5,5), cv2.BORDER_DEFAULT)
# Displaying the blurred image
plt.imshow(img)
```

INPUT :

OUTPUT :





Program 3: Implement Averaging and Gaussian filter

```
# importing libraries
import cv2
import numpy as np

image1 = cv2.imread("E:/unnati/DIP/fruits.jpg")

cv2.imshow('Original Image', image1)
cv2.waitKey(0)

# Gaussian Blur
Gaussian1 = cv2.GaussianBlur(image1, (7, 7), 0)
cv2.imshow('Gaussian Blurring', Gaussian1)
cv2.waitKey(0)

# Averaging the image
Average1 = cv2.blur(image1, (3,3))
# Displaying the blurred image
plt.imshow(Average1)

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

INPUT :

OUTPUT:
Original image :

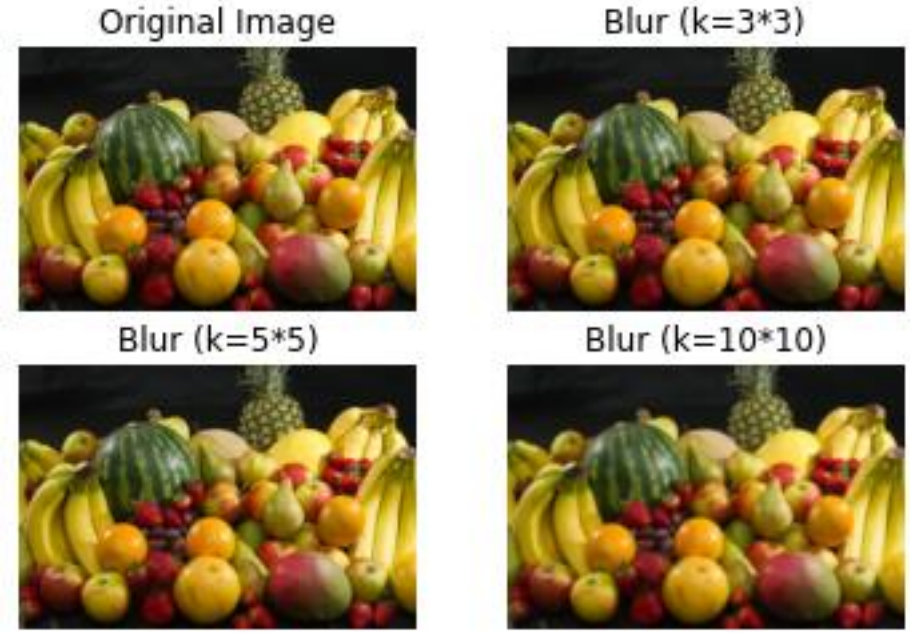
Gaussian Filter:

Averaging Filter:

Program 4: BOX BLUR/ MEAN FILTER, DISPLAY IN MATPLOT

```
import cv2
import matplotlib.pyplot as plt

# load the input image
img = cv2.imread("D:/Diana/DIPLAB/DIP1.png")

# convert BGR to RGB to display using matplotlib
imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# kernal size
ksize = (3, 3)
# Using cv2.blur() method
k3blur = cv2.blur(imgRGB, ksize)

# kernal size
ksize = (5, 5)
# Using cv2.blur() method
k5blur = cv2.blur(imgRGB, ksize)

# kernal size
ksize = (10, 10)
# Using cv2.blur() method
k10blur = cv2.blur(imgRGB, ksize)

# display Original, Grayscale and Binary Images
plt.subplot(221),plt.imshow(imgRGB),plt.title('Original Image'), plt.axis('off')
plt.subplot(222),plt.imshow(k3blur),plt.title('Blur (k=3*3)'), plt.axis('off')
plt.subplot(223),plt.imshow(k5blur),plt.title('Blur (k=5*5)'), plt.axis('off')
plt.subplot(224),plt.imshow(k10blur),plt.title('Blur (k=10*10)'), plt.axis('off')
```

OUTPUT:



Program 5: Gaussian Blur ,disolay in matplot

```
import cv2
import matplotlib.pyplot as plt

# load the input image
img = cv2.imread("D:/Diana/DIPLAB/DIP1.png")

# convert BGR to RGB to display using matplotlib
imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Apply Gaussian blur with kernel 5*5
GausImg5 = cv2.GaussianBlur(imgRGB,(5,5),5)

# Apply Gaussian blur with kernel 15*15
GausImg15 = cv2.GaussianBlur(imgRGB,(15,15),5)

#Plot Original, Blurred images
plt.subplot(131),plt.imshow(imgRGB),plt.title('Original Image'), plt.axis('off')
plt.subplot(132),plt.imshow(GausImg5),plt.title('Gaussian Blur (k=5*5)'), plt.axis('off')
plt.subplot(133),plt.imshow(GausImg15),plt.title('Gaussian Blur (k=15*15)'), plt.axis('off')
```

OUTPUT:

PROGRAM 8: Image Filter: Bilateral Filter, Box Filter, Erosion

Aim : Write a program in python to implement Bilateral Filter, Box Filter and Erosion

Image filtering is changing the appearance of an image by altering the colors of the pixels. Increasing the contrast as well as adding a variety of special effects to images are some of the results of applying filters

Filters in image processing are matrices or kernels convolved over the images to modify their appearance or extract their specific features. Filters work by modifying an image's pixel values based on neighboring pixels' values.

1. Bilateral Filter

A bilateral filter is used for smoothening images and reducing noise, while preserving edges. These Mean, Median filters often result in a loss of important edge information, since they blur out everything, irrespective of it being noise or an edge. A bilateral filter counters this problem.

It replaces the intensity of each pixel with a weighted average of intensity values from nearby pixels

Function used

cv2. bilateralFilter(src, dst, d, sigmaColor, sigmaSpace, borderType)

src − A Mat object representing the source (input image) for this operation.
dst − A Mat object representing the destination (output image) for this operation.
d − A variable of the type integer representing the diameter of the pixel neighborhood.
sigmaColor − A variable of the type integer representing the filter sigma in the color space.
sigmaSpace − A variable of the type integer representing the filter sigma in the coordinate space.
borderType − An integer object representing the type of the border used.
Return Value: a Mat object representing the blurred image

Program 1 : Implement Bilateral blur in Python

```
import cv2
import matplotlib.pyplot as plt

# load the input image
img = cv2.imread("E:/unnati/DIP/2.jpg")

# convert BGR to RGB to display using matplotlib
imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Apply Bilateral blur with d=5, sigma=75
BilImg5 = cv2.bilateralFilter(imgRGB,5, 75, 75)
```

```
# Apply Bilateral blur with d=15, sigma=100
BilImg15 = cv2.bilateralFilter(imgRGB,15, 100, 100)

plt.subplot(131),plt.imshow(imgRGB),plt.title('Original Image'), plt.axis('off')
plt.subplot(132),plt.imshow(BilImg5),plt.title('Bilateral Blur d=5'), plt.axis('off')
plt.subplot(133),plt.imshow(BilImg15),plt.title('Bilateral Blur d=15'), plt.axis('off')
```

OUTPUT:

2. Box Filtering

Box Filter is a low-pass filter that smooths the image by making each output pixel the average of the surrounding ones, removing details, noise and and edges from images

This is done by convolving the image with a normalized box filter. It simply takes the average of all the pixels under kernel area and replaces the central element with this average. This is done by the function **cv2.blur()** or **cv2.boxFilter()**. Check the docs for more details about the kernel. We should specify the width and height of kernel. A 3x3 normalized box filter would look like this:

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Program 2 : Implement Box filter in Python

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

img = cv2.imread("E:/unnati/DIP/opencv.png")

blur = cv2.blur(img,(5,5))

plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
```

```
plt.xticks([]), plt.yticks([])
plt.show()
```

OUTPUT

PROGRAM 3: COMPARISON OF BILATERAL FILTERING AND BOX FILTER, DISPLAY IN MATPLOT

```
import cv2
import matplotlib.pyplot as plt

# load the input image
img = cv2.imread("E:/unnati/DIP/images1.jpeg")

# convert BGR to RGB to display using matplotlib
imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Apply Bilateral blur with d=5, sigma=75
BilImg5 = cv2.bilateralFilter(imgRGB,5, 75, 75)

# Apply Box blur with k=5
k5blur = cv2.blur(imgRGB, (5,5))

plt.subplot(131),plt.imshow(imgRGB),plt.title('Original Image'), plt.axis('off')
plt.subplot(132),plt.imshow(BilImg5),plt.title('Bilateral Blur d=5'), plt.axis('off')
plt.subplot(133),plt.imshow(k5blur),plt.title('Box Blur k=5'), plt.axis('off')
```

OUTPUT:

Morphological Operations

- It is a broad set of image processing operations that process digital images based on their shapes.
- Each image pixel is corresponding to the value of other pixel in its neighborhood.
- By choosing the shape and size of the neighborhood pixel, we can construct a morphological operation that is sensitive to specific shapes in the input image.

Types of Morphological operations:

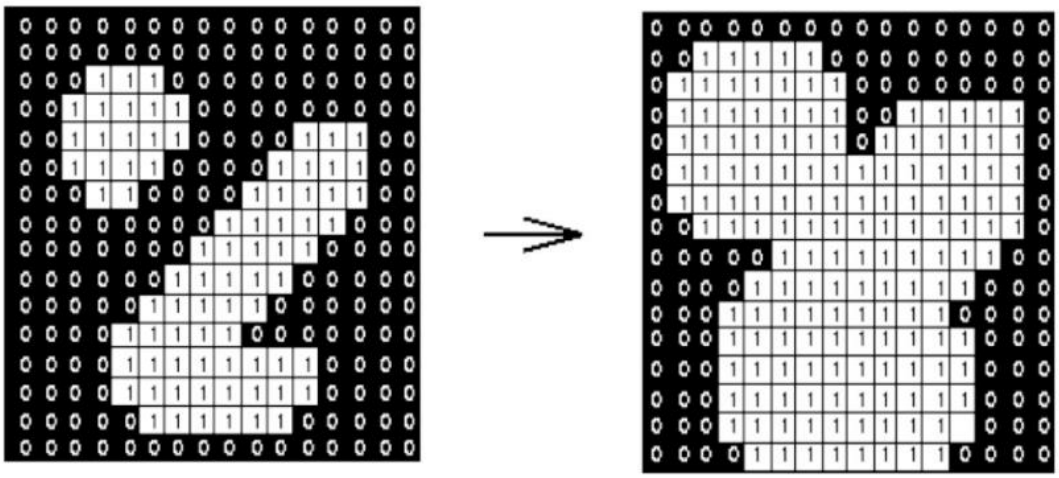
1. Dilation: Dilation adds pixels on the object boundaries.Enlarges a shape by adding pixels to the boundary of an object
2. Erosion: Erosion removes pixels on object boundaries.Removes thin lines by removing pixels from the boundary of an object
3. Open: The opening operation erodes an image and then dilates the eroded image, using the same structuring element for both operations.
4. Close: The closing operation dilates an image and then erodes the dilated image, using the same structuring element for both operations.

Dilation

- Dilation is a morphological operation that expands the boundaries of an object in an image. This is done by convolving the image with a structuring element, which determines the size and shape of the dilation.
- The output of the dilation operation is a new image where the pixels in the original image are expanded or dilated.

Structuring Element

- The structuring element is a small binary image used to define the neighborhood of a pixel during dilation.
- The shape and size of the structuring element determine the degree of dilation. Structuring elements can take different shapes, such as squares, diamonds, disks, or lines during erosion and dilation in image processing.
- The size and shape of the structuring element define how many numbers of the pixel should be added or removed from the objects in an image.
- It is a matrix of 1's and 0's. The center pixel of the image is called the origin.
- It contains an image A with some kernel (B), which can have any shape or size, generally a square or circle. Here the kernel B has a defined anchor point. It is the center of the kernel.
- In the next step, the kernel is overlapped over the image to calculate maximum pixel values. When the computation is completed, the image is replaced with an anchor at the center. The brighter areas increase in size that made increment in image size.



Effect of dilation using a 3×3 square structuring element

The dilate() function

The dilate() function in image processing libraries is used to perform dilation on an image using a structuring element.

cv2.dilate(src, dst, kernel)

Parameters: The dilate() function accepts the following argument:

src - It represents the input image.

dst - It represents the output image.

kernel - It represents the kernel.

Program:

```
import cv2
import numpy as np

# Read image
img = cv2.imread("E:/unnati/DIP/1.png", 0) // loads the image as grayscale image

# Define structuring element
kernel = np.ones((5,5),np.uint8)

# Perform dilation
dilation = cv2.dilate(img, kernel, iterations = 1)

# Display results
cv2.imshow('Original',img)
cv2.imshow('Dilation',dilation)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:

EROSION

- Erosion is much similar to dilation.
- The difference is that the pixel value calculated minimum rather than the maximum in dilation.
- The image is replaced under the anchor point with that calculated minimum pixel.
- Unlikely dilation, the regions of darker shades increase. While it decreases in white shade or brighter side.
- Erosion is a morphological operation that shrinks the boundaries of an object in an image.
- This is done by convolving the image with a structuring element, which determines the size and shape of the erosion.
- The output of the erosion operation is a new image where the pixels in the original image are eroded or shrunk.

cv2.erode(src, dst, kernel)
Parameters:
src - It represents the source(input) image.
dst - It represents the destination (output) image.
kernel - It represents the Kernel.

Program:

```
import cv2
import numpy as np
# Read input image
img = cv2.imread("E:/unnati/DIP/1.png", 0) // 0: loads the image as gray scale image
# Define structuring element
kernel = np.ones((5,5), np.uint8)
# Perform erosion
erosion = cv2.erode(img, kernel, iterations = 1)
# Display results
cv2.imshow('Input Image', img)
cv2.imshow('Erosion', erosion)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:

Dilation	Erosion
It increases the size of the objects.	It decreases the size of the objects.
It fills the holes and broken areas.	It removes the small anomalies.
It connects the areas that are separated by space smaller than structuring element.	It reduces the brightness of the bright objects.
It increases the brightness of the objects.	It removes the objects smaller than the structuring element.
Distributive, duality, translation and decomposition properties are followed.	It also follows the different properties like duality etc.
It is XOR of A and B.	It is dual of dilation.
It is used prior in Closing operation.	It is used later in Closing operation.
It is used later in Opening operation.	It is used prior in Opening operation.

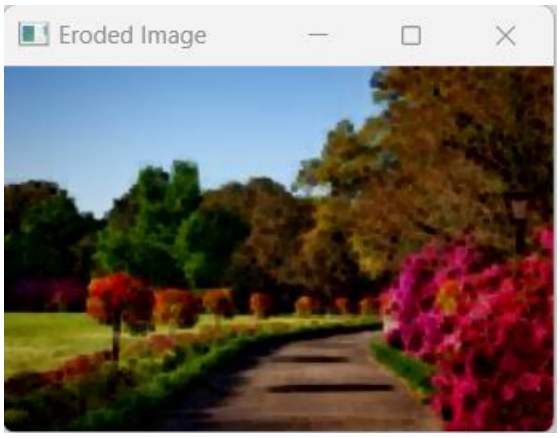
Program : To implement Erosion and Dilation

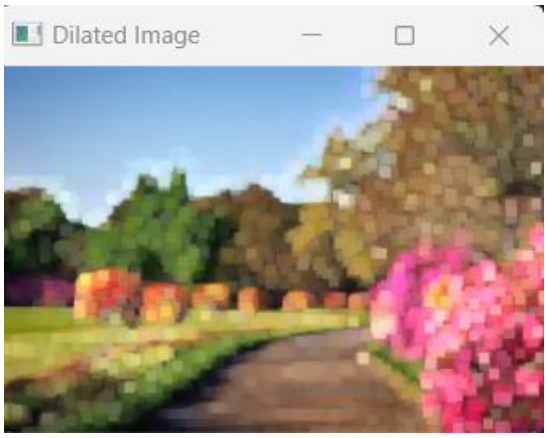
```
import cv2
import numpy as np
input_image = cv2.imread("E:/unnati/DIP/5.jpeg", 1) // Color image

// define a 3x3 matrix of ones using the np.ones() function,which will be used as the kernel for erosion.
kernel = np.ones((3,3), np.uint8)

//Perform Erosion
//specify the number of iterations as 1, which determines the extent of erosion. Higher values
//of iterations will result in a greater erosion of the image.
eroded_image = cv2.erode(input_image, kernel, iterations=1)

cv2.imshow('Input Image', input_image)
cv2.imshow('Eroded Image', eroded_image)
cv2.waitKey(0)
img = cv2.imread("E:/unnati/DIP/5.jpeg")

//defined a 5x5 rectangular structuring element using the NumPy library.
kernel = np.ones((5,5),np.uint8)
```

```
dilated_img = cv2.dilate(img, kernel, iterations=1)
cv2.imshow('Dilated Image', dilated_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:





PROGRAM 9 : THRESHOLDING – SIMPLE and ADAPTIVE

Aim :Perfrom the various thresholding techniques using OpenCV

Thresholding is a type of image segmentation, where we change the pixels of an image to make the image easier to analyze. In thresholding, we convert an image from colour or grayscale into a binary image, i.e., one that is simply black and white

Function:

(T, threshImage) = cv2.threshold(src, thresh, maxval, type)

- The first parameter is our source image, or the image that we want to perform thresholding on. This image should be grayscale.
- The second parameter, thresh, is the threshold value which is used to classify the pixel intensities in the grayscale image.
- The third parameter, maxval, is the pixel value used if any given pixel in the image passes the thresh test.
- The fourth parameter is the thresholding method to be used. The type value can be any of:

1. cv2.THRESH_BINARY
2. cv2.THRESH_BINARY_INV
3. cv2.THRESH_TRUNC
4. cv2.THRESH_TOZERO
5. cv2.THRESH_TOZERO_INV

The cv2.threshold then returns a tuple of two values. The first value, T , is the value that was used for the thresholding. The second value is our actual thresholded image.

There are various types of image thresholding and they are as follows:

1. Simple thresholding
2. Adaptive thresholding
3. Adaptive mean thresholding
4. Gaussian thresholding
5. Otsu’s thresholding

1. SIMPLE THRESHOLDING

Simple Image thresholding is a classical method of image thresholding in which pixels above and below a set threshold are assigned new values. This technique is used when there is high contrast between the background and the foreground in the given image.

T <- threshold
MaxIntensityValue<- the highest intensity value by which the pixel value can be replaced.
MinIntensityValue<- the lowest intensity value by which the pixel value can be replaced.
I(x, y) denotes the intensity value of a pixel at coordinate x, y.
The operation can be formulated as

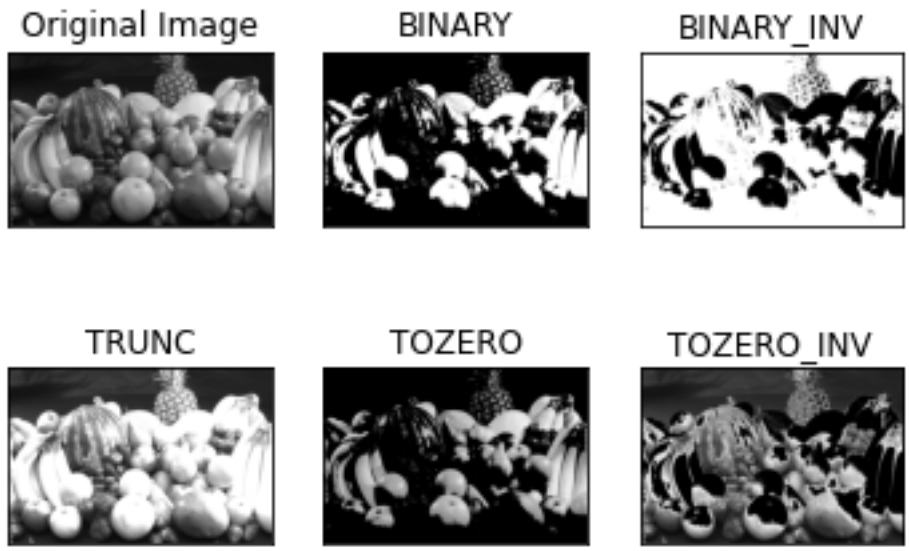
If I(x, y) < T:
I(x, y) <- MinIntensityValue
else:
I(x, y) <- MaxIntensityValue

cv2. THRESH_BINARY	This method in cv2 library sets the values below the threshold to 0 and and above threshold to 255.	If I(x, y) < T: I(x, y) <- 0 else: I(x, y) <- 255
cv2.THRESH_BINARY_INV	This method in cv2 library works the opposite way of cv2.THRESH_BINARY works	If I(x, y) < T: I(x, y) <- 255 else: I(x, y) <- 0
cv2.THRESH_TRUNC	This method in the cv2 library truncates all values above the threshold value to the threshold value without changing the rest of the pixel values.	If I(x, y) < T: I(x, y) <- I(x, y) else: I(x, y) <- T
cv2.THRESH_TOZERO	This method in the cv2 library sets the values below the threshold to 0.	If I(x, y) < T: I(x, y) <- 0 else: I(x, y) <- I(x, y)
cv2.THRESH_TOZERO_INV	This method in cv2 library works the opposite way of cv2.THRESH_TOZERO works	If I(x, y) < T: I(x, y) <- I(x, y) else: I(x, y) <- 0

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

img = cv2.imread('E:/unnati/DIP/fruits.jpg',0)
ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)

titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]

for i in range(6):
plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
plt.title(titles[i])
plt.xticks([]),plt.yticks([])

plt.show()
```

INPUT :

OUTPUT



GLOBAL THRESHOLDING

In global thresholding, each pixel value in the image is compared with a single (global) threshold value.

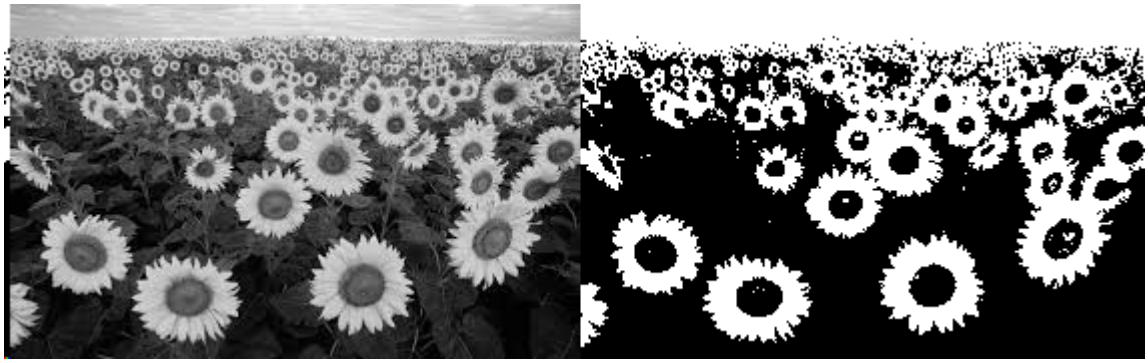
```
        if pixel_value> threshold:
        pixel_value = MAX
        else:
pixel_value = 0
```

1. Select initial threshold value, typically the mean 8-bit value of the original image.
2. Divide the original image into two portions;
    1. Pixel values that are less than or equal to the threshold; background
    2. Pixel values greater than the threshold; foreground
3. Find the average mean values of the two new images
4. Calculate the new threshold by averaging the two means.
5. If the difference between the previous threshold value and the new threshold value are below a specified limit, you are finished. Otherwise apply the new threshold to the original image keep trying.

```
import cv2
import numpy as np

def thres_finder(img, thres=20,delta_T=1.0):

    # Step-2: Divide the images in two parts
x_low, y_low = np.where(img<=thres)
x_high, y_high = np.where(img>thres)

    # Step-3: Find the mean of two parts
mean_low = np.mean(img[x_low,y_low])
mean_high = np.mean(img[x_high,y_high])

    # Step-4: Calculate the new threshold
new_thres = (mean_low + mean_high)/2

    # Step-5: Stopping criteria, otherwise iterate
    if abs(new_thres-thres)<delta_T:
        return new_thres
    else:
        return thres_finder(img, thres=new_thres,delta_T=1.0)

# Load an image in the greyscale
img = cv2.imread('E:/unnati/DIP/img1.jpg',cv2.IMREAD_GRAYSCALE)
# apply threshold finder
vv1 = thres_finder(img, thres=30,delta_T=1.0)
# threshold the image
ret, thresh = cv2.threshold(img,vv1,255,cv2.THRESH_BINARY)
# Display the image side by side
out = cv2.hconcat([img,thresh])
cv2.imshow('threshold',out)
cv2.waitKey(0)
```

INPUT

OUTPUT



ADAPTIVE THRESHOLDING

Adaptive thresholding is also often called dynamic or local thresholding technique. It is applied for the cases where the lighting conditions are different in different regions of the image and the threshold value is calculated for smaller regions.

The threshold value for a pixel is determined on the basis of the region around it. Different threshold values are obtained for the different regions of the same image. Adaptive thresholding at pixel level yields superior results as compared to global thresholding, particularly in cases where the image has regions of varying levels of contrasts.

In OpenCV, cv2.adaptiveThreshold() is used for adaptive thresholding.

The types of adaptive thresholding are:

1. cv2.ADAPTIVE_THRESH_MEAN_C: Where threshold value = (Mean of the neighboring values – constant. It is the mean of the block size of a pixel neighborhood subtracted by the constant value.

2. cv2.ADAPTIVE_THRESH_GAUSSIAN_C: Where threshold Value = Gaussian-weighted sum of the neighboring values – constant. It is a weighted sum of the block size of a pixel neighborhood subtracted by a constant value.

THRESH_BINARY

$$dst(x,y) = \begin{cases} maxValue & \text{if } src(x,y) > T(x,y) \\ 0 & \text{otherwise} \end{cases}$$

THRESH_BINARY_INV

$$dst(x,y) = \begin{cases} 0 & \text{if } src(x,y) > T(x,y) \\ maxValue & \text{otherwise} \end{cases}$$

cv2.adaptiveThreshold(src, maxVal, adaptiveMethod, thresholdType, blockSize, C)

Parameters

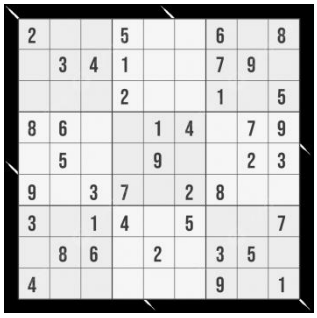
- src: The source image, which first has to be converted to grayscale color space.
- maxVal: The maximum value that pixels exceeding the threshold value can take.
- adaptiveMethod: Type of adaptive thresholding method to be used
- thresholdType: Type of adaptive thresholding to be applied.
- blockSize: Size of a neighborhood of the pixel used to calculate the threshold value.
- constant: A constant value that is subtracted from the mean or weighted mean.

PROGRAM

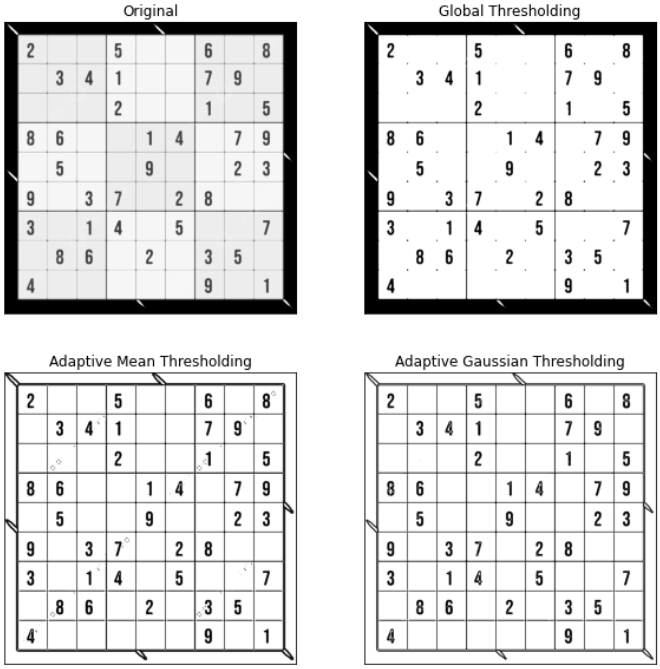
```
# Importing OpenCV
import cv2
# Importing matplotlib.pyplot
import matplotlib.pyplot as plt
# Reading the image
img = cv2.imread(r'E:/unnati/DIP/sudoku.png')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# Displaying the image
plt.imshow(img)
# Converting to grayscale color space and applying median blur
img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
img = cv2.medianBlur(img,5)
# Global thresholding
ret,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
# Adaptive thresholding
th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY,11,2)
th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY,11,2)
# Plotting the images using matplotlib
titles = ['Original', 'Global Thresholding',
          'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']
images = [img, th1, th2, th3]
# Specifying the grid size
plt.figure(figsize=(10,10))
# Number of images in the grid 2*2 = 4
```

```
for i in range(4):
plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
plt.title(titles[i])
plt.xticks([]),plt.yticks([])

# Displaying the grid
plt.show()
```

INPUT

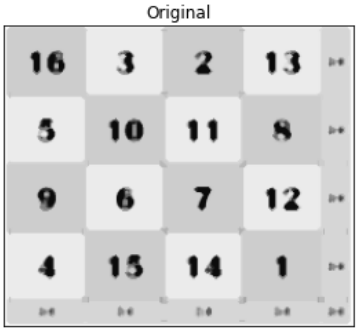


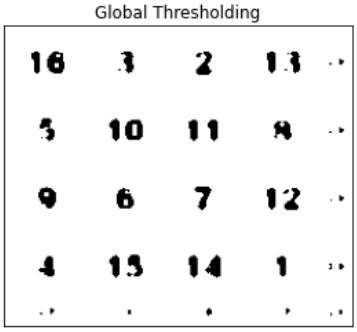
OUTPUT

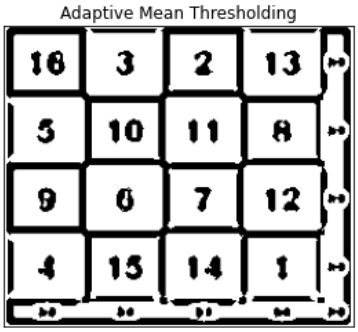


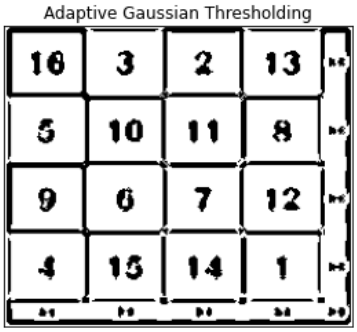
INPUT

OUTPUT


Original
16 3 2 13
5 10 11 8
9 6 7 12
4 15 14 1


Global Thresholding


Adaptive Mean Thresholding
16 3 2 13
5 10 11 8
9 6 7 12
4 15 14 1


Adaptive Gaussian Thresholding
16 3 2 13
5 10 11 8
9 6 7 12
4 15 14 1

PROGRAM 10 : SOBEL OPERATOR

Aim : Implement Sobel Operator using OpenCV

Edge detection is one of the fundamental operations when we perform image processing. It helps us reduce the amount of data (pixels) to process and maintains the structural aspect of the image.

1. The gradient (Sobel - first order derivatives) based edge detector
2. The Laplacian (2nd order derivative, so it is extremely sensitive to noise) based edge detector.

Sobel Edge Detection
Sobel edge detector is a gradient based method based on the first order derivatives.
It calculates the first derivatives of the image separately for the X and Y axes.
The operator uses two 3X3 kernels which are convolved with the original image to calculate approximations of the derivatives - one for horizontal changes, and one for vertical.
The horizontal Sobel filter highlights edges that run from left to right, while the vertical Sobel filter highlights edges that run from top to bottom. The below matrices Gx and Gy denotes show the Sobel operator in the x and y direction respectively.

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

The filter is used to calculate the gradient of the image at that pixel location.
The gradient of an image is a measure of how much the intensity of the image changes at that point.
If the gradient is high, it means there is a strong change in intensity, which indicates an edge in the image.

Sobel(src, dst, ddepth, dx, dy)

Parameters:

- src − An object of the class Mat representing the source (input) image.
- dst − An object of the class Mat representing the destination (output) image.
- ddepth − An integer variable representing the depth of the image (-1)
- dx − An integer variable representing the x-derivative. (0 or 1)
- dy − An integer variable representing the y-derivative. (0 or 1)

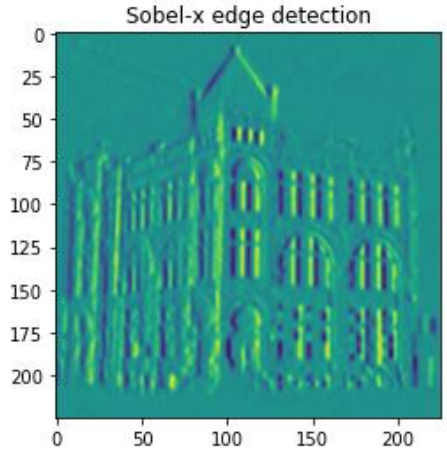
The Sobel filter is applied to an image by convolving the filter with each pixel of the image. The result of the convolution is a new image that highlights the edges in the original image. This new image is called the gradient image, and it represents the magnitude of the gradient at each pixel location.
the Sobel filter-based edge detection technique uses a threshold to convert the gradient image into a binary image, where each pixel is either white or black. The threshold is chosen based on the desired level of sensitivity for detecting edges in the image. The resulting binary image

is a map of the edges in the original image, which can be used for further analysis and processing in computer vision applications.

```
import cv2
import matplotlib.pyplot as plt
# Read the original image
img = cv2.imread('E:/unnati/DIP/img11.jpeg')
# converting because opencv uses BGR as default
RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(RGB_img)
# converting to gray scale
gray = cv2.cvtColor(RGB_img, cv2.COLOR_BGR2GRAY)
# remove noise
img = cv2.GaussianBlur(gray,(3,3),0)
# convolute with sobel kernels
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)  # x
sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)  # y
#Plotting images
plt.imshow(sobelx)
plt.title("Sobel-x edge detection")
```

INPUT

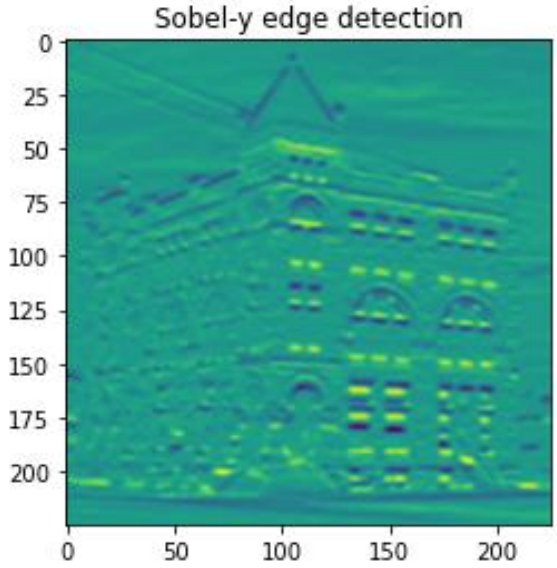
OUTPUT : Sobel -x edge detection



Add the below code to the above program for y detection

```
plt.imshow(sobely)
plt.title("Sobel-y edge detection")
```

OUTPUT



Laplacian Edge Detection

Unlike the Sobel edge detector, the Laplacian edge detector uses only one kernel. It calculates second order derivatives in a single pass.

A kernel used in this Laplacian detection looks like this:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

If we want to consider the diagonals, we can use the kernel below:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

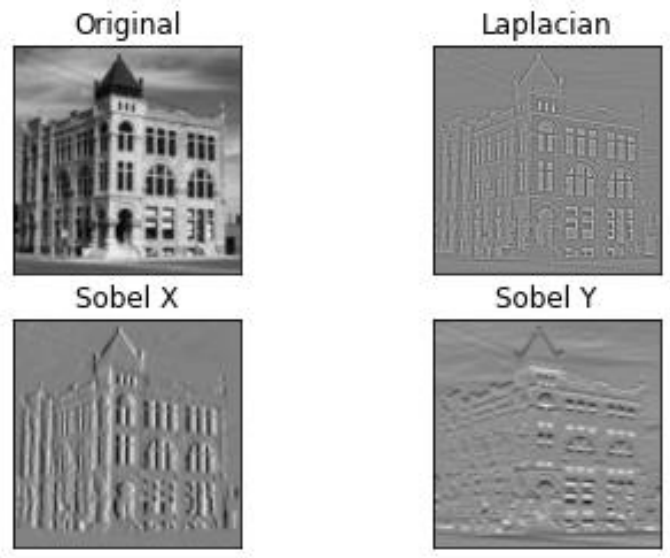
cv2.Laplacian(src, ddepth, other_options...)

where ddepth is the desired depth of the destination image.

PROGRAM

```
import cv2

from matplotlib import pyplot as plt

# loading image
#img0 = cv2.imread('SanFrancisco.jpg',)
img0 = cv2.imread('E:/unnati/DIP/img11.jpeg',)

# converting to gray scale
gray = cv2.cvtColor(img0, cv2.COLOR_BGR2GRAY)

# remove noise
img = cv2.GaussianBlur(gray,(3,3),0)

# convolute with proper kernels
laplacian = cv2.Laplacian(img,cv2.CV_64F)
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)  # x
sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)  # y

plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray')
plt.title('Laplacian'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray')
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])

plt.show()
```

OUTPUT

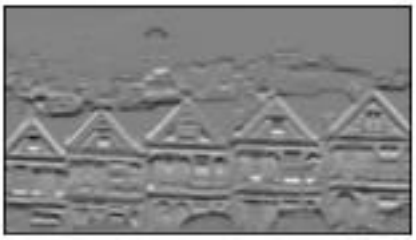


Original

Laplacian

Sobel X

Sobel Y



Original

Laplacian

Sobel X

Sobel Y

PROGRAM 10 : SOBEL OPERATOR

Aim : Implement Sobel Operator using OpenCV

Edge detection is one of the fundamental operations when we perform image processing. It helps us reduce the amount of data (pixels) to process and maintains the structural aspect of the image.

3. The gradient (Sobel - first order derivatives) based edge detector
4. The Laplacian (2nd order derivative, so it is extremely sensitive to noise) based edge detector.

Sobel Edge Detection
Sobel edge detector is a gradient based method based on the first order derivatives.
It calculates the first derivatives of the image separately for the X and Y axes.
The operator uses two 3X3 kernels which are convolved with the original image to calculate approximations of the derivatives - one for horizontal changes, and one for vertical.
The horizontal Sobel filter highlights edges that run from left to right, while the vertical Sobel filter highlights edges that run from top to bottom. The below matrices Gx and Gy denotes show the Sobel operator in the x and y direction respectively.

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

The filter is used to calculate the gradient of the image at that pixel location.
The gradient of an image is a measure of how much the intensity of the image changes at that point.
If the gradient is high, it means there is a strong change in intensity, which indicates an edge in the image.

Sobel(src, dst, ddepth, dx, dy)

Parameters:

- src − An object of the class Mat representing the source (input) image.
- dst − An object of the class Mat representing the destination (output) image.
- ddepth − An integer variable representing the depth of the image (-1)
- dx − An integer variable representing the x-derivative. (0 or 1)
- dy − An integer variable representing the y-derivative. (0 or 1)

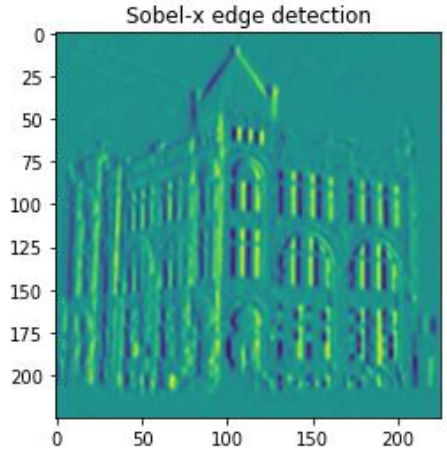
The Sobel filter is applied to an image by convolving the filter with each pixel of the image. The result of the convolution is a new image that highlights the edges in the original image. This new image is called the gradient image, and it represents the magnitude of the gradient at each pixel location.
the Sobel filter-based edge detection technique uses a threshold to convert the gradient image into a binary image, where each pixel is either white or black. The threshold is chosen based on the desired level of sensitivity for detecting edges in the image. The resulting binary image

is a map of the edges in the original image, which can be used for further analysis and processing in computer vision applications.

```
import cv2
import matplotlib.pyplot as plt
# Read the original image
img = cv2.imread('E:/unnati/DIP/img11.jpeg')
# converting because opencv uses BGR as default
RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(RGB_img)
# converting to gray scale
gray = cv2.cvtColor(RGB_img, cv2.COLOR_BGR2GRAY)
# remove noise
img = cv2.GaussianBlur(gray,(3,3),0)
# convolute with sobel kernels
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)  # x
sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)  # y
#Plotting images
plt.imshow(sobelx)
plt.title("Sobel-x edge detection")
```

INPUT

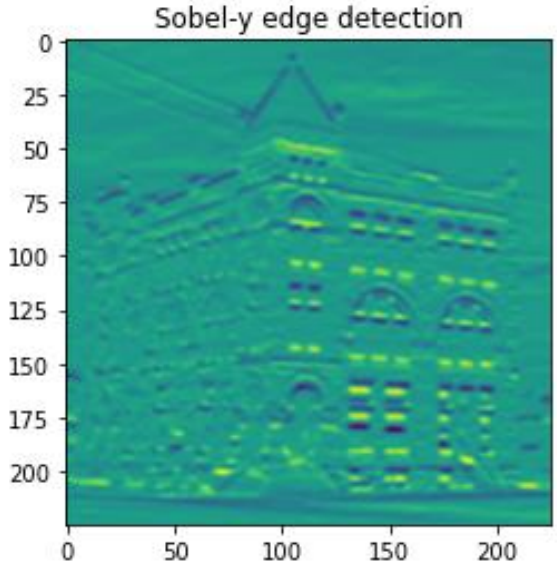
OUTPUT : Sobel -x edge detection



Add the below code to the above program for y detection

```
plt.imshow(sobely)
plt.title("Sobel-y edge detection")
```

OUTPUT



Laplacian Edge Detection

Unlike the Sobel edge detector, the Laplacian edge detector uses only one kernel. It calculates second order derivatives in a single pass.

A kernel used in this Laplacian detection looks like this:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

If we want to consider the diagonals, we can use the kernel below:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

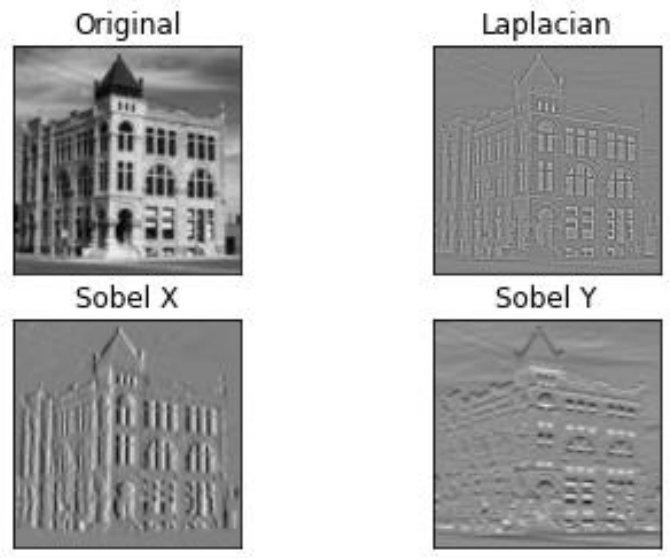
cv2.Laplacian(src, ddepth, other_options...)

where ddepth is the desired depth of the destination image.

PROGRAM

```
import cv2

from matplotlib import pyplot as plt

# loading image
#img0 = cv2.imread('SanFrancisco.jpg',)
img0 = cv2.imread('E:/unnati/DIP/img11.jpeg',)

# converting to gray scale
gray = cv2.cvtColor(img0, cv2.COLOR_BGR2GRAY)

# remove noise
img = cv2.GaussianBlur(gray,(3,3),0)

# convolute with proper kernels
laplacian = cv2.Laplacian(img,cv2.CV_64F)
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)  # x
sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)  # y

plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray')
plt.title('Laplacian'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray')
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])

plt.show()
```

OUTPUT

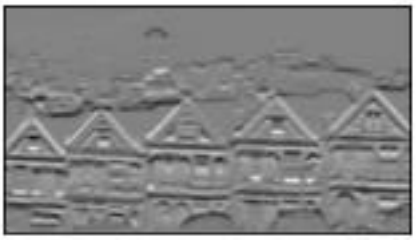


Original

Laplacian

Sobel X

Sobel Y



Original

Laplacian

Sobel X

Sobel Y

PROGRAM 11 : DFT, DCT, DWT

Aim : Implement DFT, DCT, DWT using OpenCV

The Fourier Transform is a mathematical tool used to decompose a signal into its frequency components. In the case of image processing, the Fourier Transform can be used to analyze the frequency content of an image, which can be useful for tasks such as image filtering

DFT (Discrete Fourier Transform)

In image processing, DFT is used for image compression and enhancement.
Transform an image into the frequency domain, apply various filters to remove noise and unwanted information, and transform it back into the spatial domain for display.
The DFT is the sampled Fourier Transform and therefore does not contain all frequencies forming an image, but only a set of samples which is large enough to fully describe the spatial domain image.
The number of frequencies corresponds to the number of pixels in the spatial domain image, i.e. the image in the spatial and Fourier domain are of the same size.

For a square image of size N×N, the two-dimensional DFT is given by:

$$F(k,l) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j)\, e^{-\imath 2\pi (\frac{ki}{N} + \frac{lj}{N})}$$

where f(a,b) is the image in the spatial domain and the exponential term is the basis function corresponding to each point F(k,l) in the Fourier space. The equation can be interpreted as: the value of each point F(k,l) is obtained by multiplying the spatial image with the corresponding base function and summing the result.

Functions Used

1. cv2.dft(src,dst,flags)

- src→input image
- dst→output array whose size and type depends on the flags.
- flags →transformation flags, representing a combination of the cv.DftFlags

2.cv2.magnitude(InputArray x, InputArray y, OutputArray magnitude)
Calculates the magnitude of 2D vectors.
The function cv::magnitude calculates the magnitude of 2D vectors formed from the corresponding elements of x and y arrays:

$$\texttt{dst}(I) = \sqrt{\texttt{x}(I)^2 + \texttt{y}(I)^2}$$

- x →floating-point array of x-coordinates of the vectors.
- y →floating-point array of y-coordinates of the vectors; it must have the same size as x.
- magnitude→output array of the same size and type as x.

3. cv2.normalize(src,dst, alpha, beta, normalization_type,dtype)
Normalize an image in OpenCV.
 src→ input image.
 dst→output array whose size and type depends on the flags.
 alpha →represents the lower range boundary value.
 beta →represents the upper range boundary value.
 normalization_type→represents the type of normalization.
 dtype→ output data encoding type

4. cv2.idft(src,dst,flags)
Returns two channels.
First channel will have the real part of the result and second channel will have the imaginary part of the result. The input image should be converted to np.
Performs an inverse Discrete Fourier transform of 1D or 2D floating-point array.
 src→input image.
 dst→ output array whose size and type depends on the flags.
 flags →transformation flags, representing a combination of the cv.DftFlags

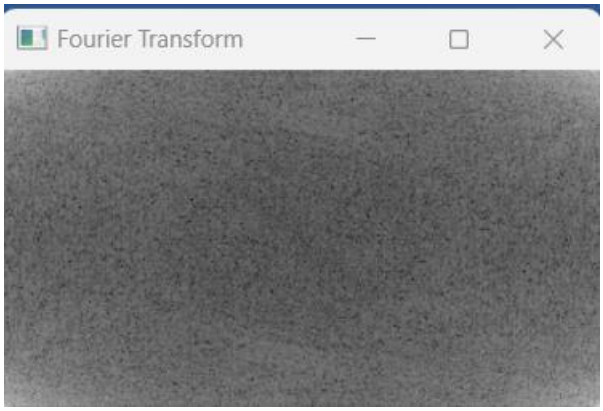
Program to find DFT and Inverse DFT

```
import cv2
import numpy as np

# load the image and convert to grayscale
image = cv2.imread("E:/unnati/DIP/download.jpeg")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Compute the Discrete Fourier Transform of the image
#Convert numpy array to float32
#cv2.dft() function --3-dimensional numpy array of shape
#the output of 2-D Fourier Transform is a 2-dimensional complex array
#the first and second channel of f are the real part and imaginary part respectively
# f_complex --> the complex version of f.

fourier = cv2.dft(np.float32(gray), flags=cv2.DFT_COMPLEX_OUTPUT)

# calculate the magnitude of the Fourier Transform
#take the log of absolute value as it has wide range
magnitude = 20*np.log(cv2.magnitude(fourier[:,:,0],fourier[:,:,1]))

# Scale the magnitude for display
#Constarint the range from 0 to 255
magnitude = cv2.normalize(magnitude, None, 0, 255, cv2.NORM_MINMAX,
cv2.CV_8UC1)

#Inverse DFT
idft = cv2.idft(fourier)
```

```
# calculate the magnitude of the Inverse Fourier Transform
imag = 20*np.log(cv2.magnitude(idft[:,:,0],idft[:,:,1]))

# Scale the magnitude for display
imagnorm = cv2.normalize(imag, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8UC1)

# Display the magnitude of the Fourier Transform
cv2.imshow("Original Image",gray)
cv2.imshow('Fourier Transform', magnitude)
cv2.imshow('Magnitude Spectrum', imagnorm)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT

OUTPUT



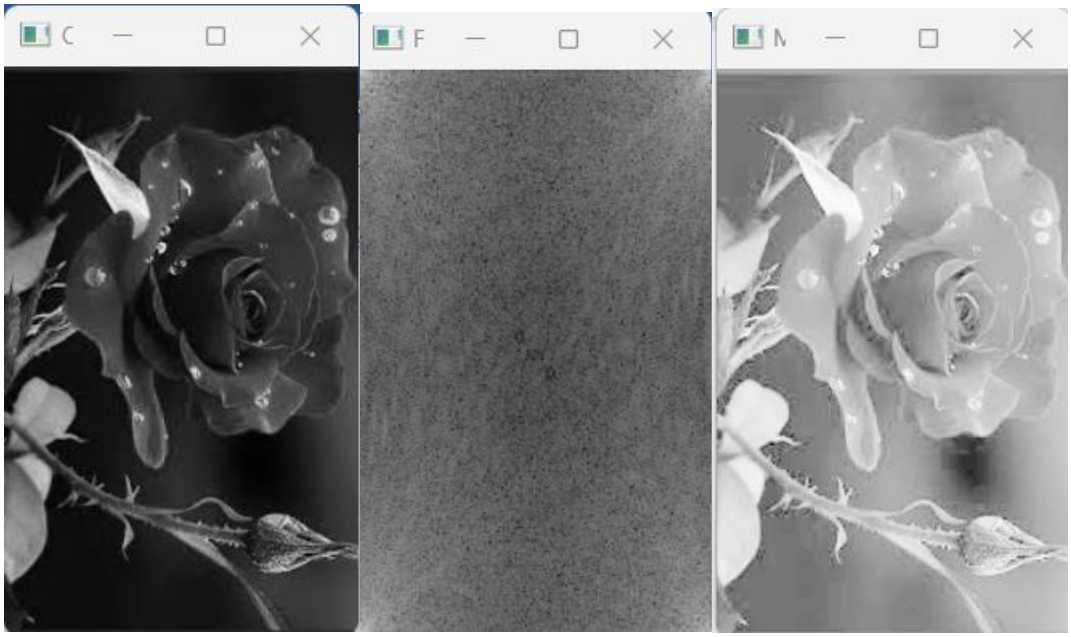


IMAGE TRANSFORMS - DISCRETE COSINE TRANSFORMS

Discrete Cosine Transform is used in lossy image compression because it has very strong energy compaction, i.e., its large amount of information is stored in very low frequency component of a

signal and rest other frequency having very small data which can be stored by using very less number of bits (usually, at most 2 or 3 bit).
The discrete cosine transform (DCT) helps separate the image into parts (or spectral sub-bands) of differing importance (with respect to the image's visual quality).
The DCT is similar to the discrete Fourier transform: it transforms a signal or image from the spatial domain to the frequency domain

Functions Used
1. cv2.dct(src,dst,flags)
Src→ input aimage
Dst→ output array whose size and type depends on the flags.
Flags→ transformation flags, representing a combination of the cv.DctFlags

2. cv2.idft(src,dst,flags)
Src→input aimage
Dst→output array whose size and type depends on the flags.
Flags→transformation flags, representing a combination of the cv.DctFlags

STEPS :

1. Import the required libraries OpenCV and NumPy. Make sure you have already installed them.

2. Read the input image using cv2.imread() method. Specify the full path of the image. Convert the input image to grayscale image using cv2.cvtColor() method. Convert the grayscale image to np.float32.

3. Find the discrete cosine transform of the image using cv2.dct(). This method takes a grayscale image in floating point. Pass flag cv2.DCT_INVERSE or cv2.DCT_ROWS to the cv2.dct() function. Visualize the discrete transform of the input image using cv2.imshow() method.

4. To visualize the input image back after the discrete cosine transform, apply inverse discrete cosine transform cv2.idct(). And convert the image to np.uint8.

   Program:

```
import cv2
import numpy as np
# load the image and convert to grayscale
image = cv2.imread("E:/unnati/DIP/rose.jpeg")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Compute the Discrete Cosine Transform of the image
dctimg = cv2.dct(np.float32(gray), cv2.DCT_INVERSE)

#Inverse DCT
idctimg = cv2.idct(dctimg)

# convert to uint8
idct = np.uint8(idctimg)
```

```
# Display the results
cv2.imshow("Original Image",gray)
cv2.imshow('Cosine Transform', dctimg)
cv2.imshow('Inverse Cosine Transform', idct)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT

OUTPUT

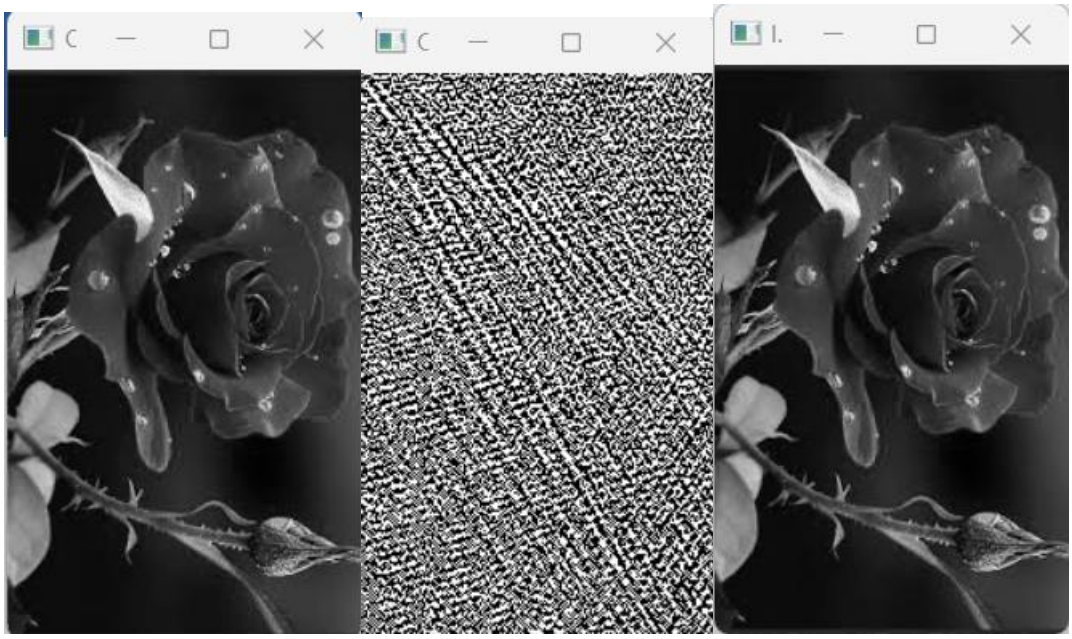


IMAGE TRANSFORMS - DISCRETE WAVELET TRANSFORMS

Discrete Wavelet Transform is a technique to transform image pixels into wavelets, which are then used for wavelet-based compression and coding

Wavelets are a mathematical tool that can be used to analyze signals and data in various fields.

A discrete wavelet transform (DWT) is a technique for transforming data into wavelet coefficients, which provide information about the time and frequency distribution of the signal.

In image processing, wavelets are used for compression, edge detection, and texture analysis.
Wavelet-based image compression techniques are widely used in digital image and video compression standards, such as JPEG2000 and MPEG4.
Wavelets can also be used for edge detection, which is the process of identifying the boundaries between different regions in an image.
Texture analysis involves analyzing the patterns of pixels in an image, and wavelets can be used to extract features from these patterns.

PyWavelets is open source wavelet transform software for Python. It combines a simple high level interface with low level C and Cython performance.
INSTALL: pip install PyWavelets or easy_installPyWavelets.

Function Used

pywt.dwt2(data, wavelet, mode='symmetric', axes=(-2, -1))

2D Discrete Wavelet Transform.
Parameters
1. dataarray_like :2D array with input data
2. wavelet :Wavelet object or name string, or 2-tuple of wavelets
Wavelet to use. This can also be a tuple containing a wavelet to apply along each axis in axes.
3. mode :str or 2-tuple of strings, optional
Signal extension mode. This can also be a tuple of modes specifying the mode to use on each axis in axes.
4. axes :2-tuple of ints, optional
Axes over which to compute the DWT. Repeated elements mean the DWT will be performed multiple times along these axes.

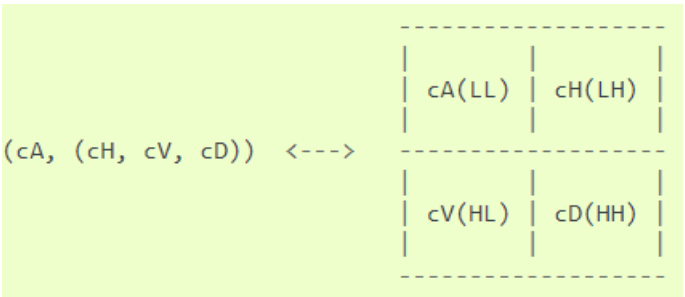
Returns

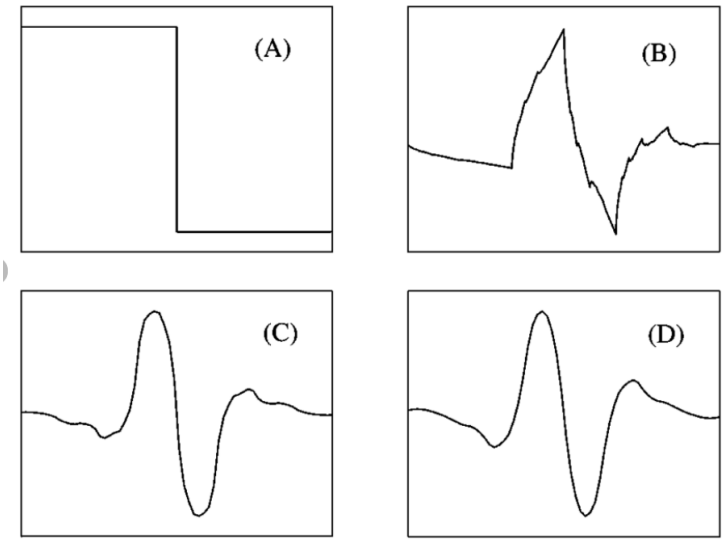
(cA, (cH, cV, cD))tuple

Approximation, horizontal detail, vertical detail and diagonal detail coefficients respectively.

Horizontal refers to array axis 0 (or axes[0] for user-specified axes).

The relation to the other common data layout where all the approximation and details coefficients are stored in one big 2D array is as follows:



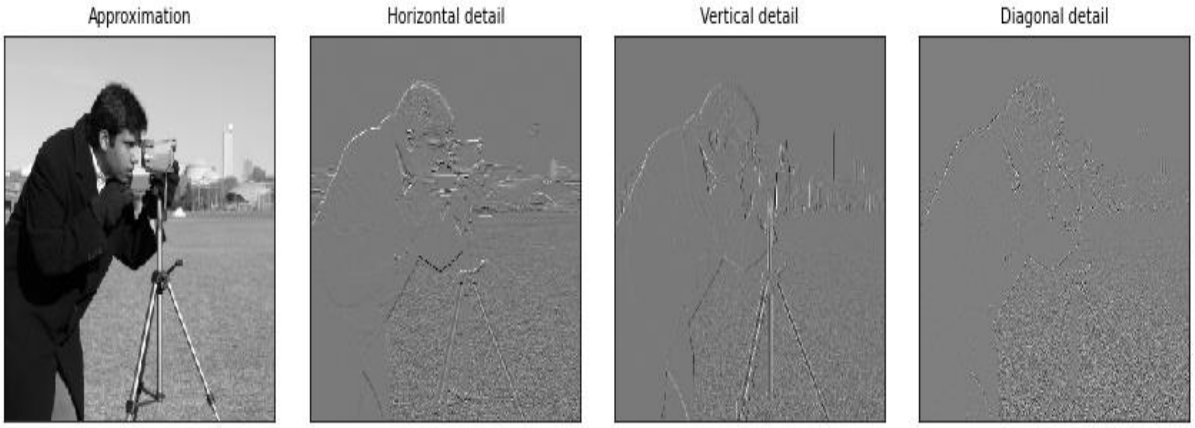


Wavelets of four different wavelet families: (A) Haar, (B) db2, (C) bior1.3, and (D) bior1.5.

PROGRAM

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

import pywt
import pywt.data


# Load image
original = pywt.data.camera()

# Wavelet transform of image, and plot approximation and details
titles = ['Approximation', ' Horizontal detail',
          'Vertical detail', 'Diagonal detail']
coeffs2 = pywt.dwt2(original, 'haar')
LL, (LH, HL, HH) = coeffs2
fig = plt.figure(figsize=(12, 3))
for i, a in enumerate([LL, LH, HL, HH]):
ax = fig.add_subplot(1, 4, i + 1)
ax.imshow(a, interpolation="nearest", cmap=plt.cm.gray)
ax.set_title(titles[i], fontsize=10)
ax.set_xticks([])
ax.set_yticks([])

fig.tight_layout()
plt.show()
```

OUTPUT:


Approximation
Horizontal detail
Vertical detail
Diagonal detail

PROGRAM 12: EDGE DETECTION

Aim: Implement Edge Detection using OpenCV

Edge detection is a technique of image processing used to identify points in a digital image with discontinuities, simply to say, sharp changes in the image brightness.
These points where the image brightness varies sharply are called the edges (or boundaries) of the image.

Edges are significant local changes of intensity in a digital image.
An edge can be defined as a set of connected pixels that forms a boundary between two disjoint regions. There are three types of edges:

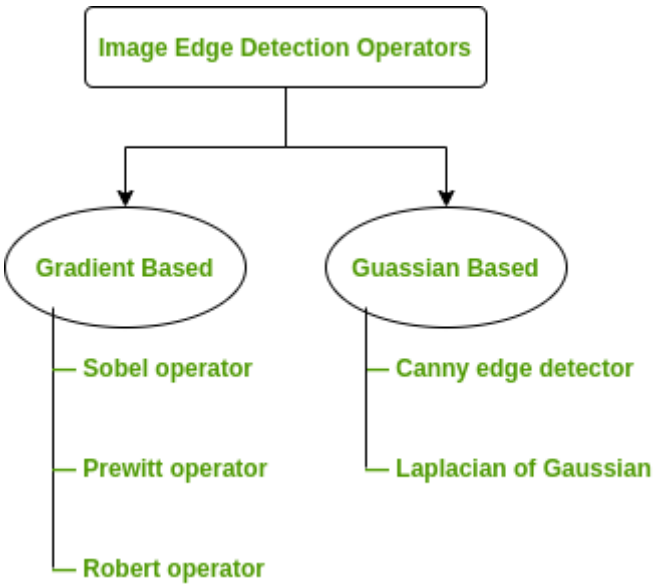
- Horizontal edges
- Vertical edges
- Diagonal edges

Edge Detection is a method of segmenting an image into regions of discontinuity. It is a widely used technique in digital image processing like

- pattern recognition
- image morphology
- feature extraction

Edge Detection Operators are of two types:

- Gradient – based operator which computes first-order derivations in a digital image like, Sobel operator, Prewitt operator, Robert operator
- Gaussian – based operator which computes second-order derivations in a digital image like, Canny edge detector, Laplacian of Gaussian



1. LAPLACIAN OPERATOR

Laplacian edge detector compares the second derivatives of an image.
It measures the rate at which the first derivative changes in a single pass.
The major difference between Laplacian and other operators like Prewitt, Sobel, Robinson and Kirsch is that these all are first order derivative masks but Laplacian is a second order derivative mask.
In this mask we have two further classifications one is Positive Laplacian Operator and other is Negative Laplacian Operator.

0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

2. SOBEL OPERATOR

The Sobel filter is a 3×3 matrix that is applied to each pixel of an image.
The filter is used to calculate the gradient of the image at that pixel location.
The gradient of an image is a measure of how much the intensity of the image changes at that point.
If the gradient is high, it means there is a strong change in intensity, which indicates an edge in the image.
There are two Sobel filters: one for detecting edges in the horizontal direction and one for detecting edges in the vertical direction. The horizontal Sobel filter highlights edges that run from left to right, while the vertical Sobel filter highlights edges that run from top to bottom.
The below matrices Gx and Gy denotes the Sobel operator in the x and y direction respectively

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

where A denotes the source image * represents the 2-dimensional convolution operation.

Functions Used

1. cv2.laplacian(src,ddepth)
o src input image
o ddepth -Depth of the destination image

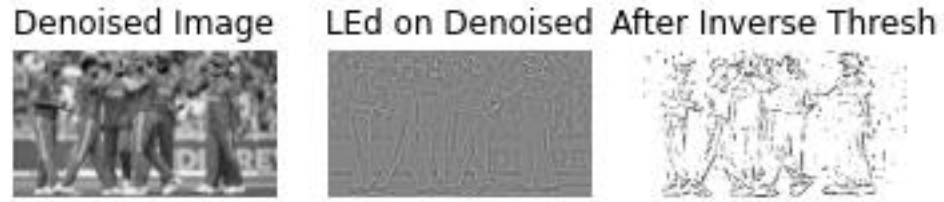
2. cv2.Sobel(src,ddepth,xorder,yorder,ksize)
o src input image
o ddepth -Depth of the destination image
o order of derivative of x (dx)
o order of derivative of y (dy)
o ksize - Kernel Size for the convolution

Program :LAPLACIAN EDGE DETECTION

```
import cv2
import matplotlib.pyplot as plt
# load the image and convert to grayscale
inpimage = cv2.imread("E:/unnati/DIP/download.jpeg")
gray = cv2.cvtColor(inpimage, cv2.COLOR_BGR2GRAY)
#Apply Laplacian Edge Detection
laplacian1 = cv2.Laplacian(gray,cv2.CV_64F)
#Apply Inverse Thresholding Laplacian Edge Detection
ret,thresh1 = cv2.threshold(laplacian1,20,255, cv2.THRESH_BINARY_INV)
#Denoising image before edge detection
denoiseimg = cv2.GaussianBlur(gray,(3,3),0)
#Apply Laplacian Edge Detection
laplacian2 = cv2.Laplacian(denoiseimg,cv2.CV_64F)
#Apply Inverse Thresholding Laplacian Edge Detection
ret,thresh2 = cv2.threshold(laplacian2,20,255, cv2.THRESH_BINARY_INV)
plt.subplot(231),plt.imshow(gray,cmap = 'gray'),plt.title('Grayscale Image'), plt.axis('off')
plt.subplot(232),plt.imshow(laplacian1,cmap = 'gray'),plt.title('Laplacian'), plt.axis('off')
plt.subplot(233),plt.imshow(thresh1,cmap = 'gray'),plt.title('After Inverse Thresh'),
plt.axis('off')
plt.subplot(234),plt.imshow(denoiseimg,cmap = 'gray'),plt.title('Denoised Image'),
plt.axis('off')
plt.subplot(235),plt.imshow(laplacian2,cmap = 'gray'),plt.title('LEd on Denoised'),
plt.axis('off')
plt.subplot(236),plt.imshow(thresh2,cmap = 'gray'),plt.title('After Inverse Thresh'),
plt.axis('off')
plt.show()
```

INPUT:

OUTPUT:



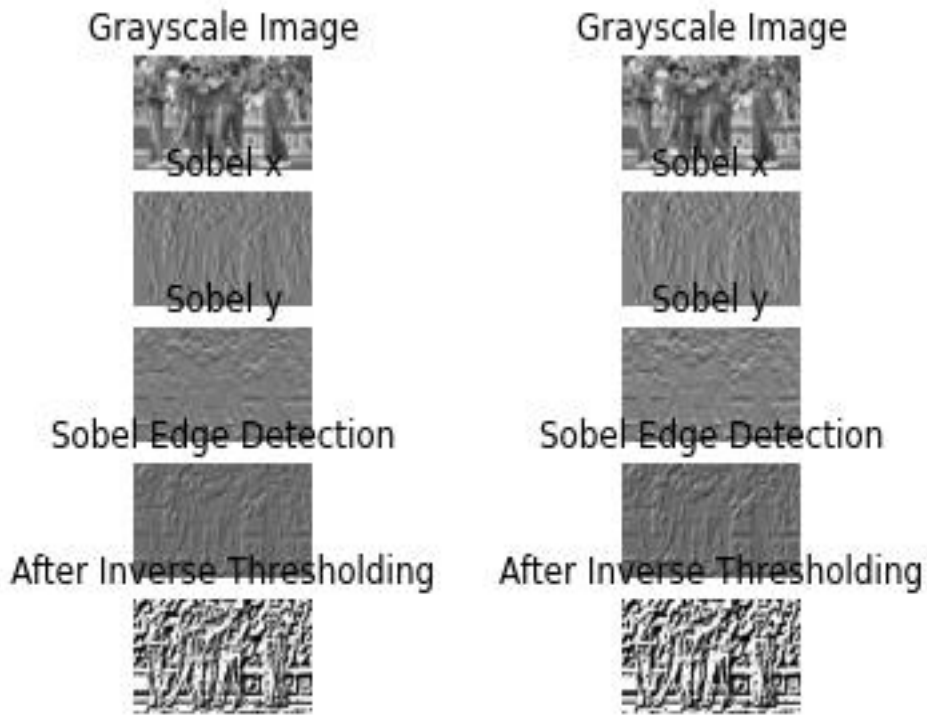
PROGRAM: SOBEL EDGE DETECTION

```
import cv2
import matplotlib.pyplot as plt
# load the image and convert to grayscale
inpimage = cv2.imread("E:/unnati/DIP/download.jpeg")
gray = cv2.cvtColor(inpimage, cv2.COLOR_BGR2GRAY)
#Apply Sobel Edge Detection
sobelx1 = cv2.Sobel(gray,cv2.CV_64F,1,0,ksize=5) # x
sobely1 = cv2.Sobel(gray,cv2.CV_64F,0,1,ksize=5) # y
#Add gradient x and Gradient y
sobel1 = sobelx1 + sobely1
#Apply Inverse Thresholding to Sobel Edge Detection
ret,thresh1 = cv2.threshold(sobel1,120,255, cv2.THRESH_BINARY_INV)
#Denoising image before edge detection
denoiseimg = cv2.GaussianBlur(gray,(3,3),0)
#Apply Sobel Edge Detection
sobelx2 = cv2.Sobel(denoiseimg,cv2.CV_64F,1,0,ksize=5) # x
sobely2 = cv2.Sobel(denoiseimg,cv2.CV_64F,0,1,ksize=5) # y
#Add gradient x and Gradient y
sobel2 = sobelx2 + sobely2
#Apply Inverse Thresholding to Sobel Edge Detection
ret,thresh2 = cv2.threshold(sobel2,120,255, cv2.THRESH_BINARY_INV)
plt.subplot2grid((5,2),(0,0)),plt.imshow(gray,cmap = 'gray'),plt.title('Grayscale Image'),
plt.axis('off')
plt.subplot2grid((5,2),(1,0)),plt.imshow(sobelx1,cmap = 'gray'),plt.title('Sobel x'),
plt.axis('off')
plt.subplot2grid((5,2),(2,0)),plt.imshow(sobely1,cmap = 'gray'),plt.title('Sobel y'),
plt.axis('off')
plt.subplot2grid((5,2),(3,0)),plt.imshow(sobel1,cmap = 'gray'),plt.title('Sobel Edge
Detection'), plt.axis('off')
plt.subplot2grid((5,2),(4,0)),plt.imshow(thresh1,cmap = 'gray'),plt.title('After Inverse
Thresholding'), plt.axis('off')
```

```
plt.subplot2grid((5,2),(0,1)),plt.imshow(gray,cmap = 'gray'),plt.title('Grayscale Image'),
plt.axis('off')
plt.subplot2grid((5,2),(1,1)),plt.imshow(sobelx2,cmap = 'gray'),plt.title('Sobel x'),
plt.axis('off')
plt.subplot2grid((5,2),(2,1)),plt.imshow(sobely2,cmap = 'gray'),plt.title('Sobel y'),
plt.axis('off')
plt.subplot2grid((5,2),(3,1)),plt.imshow(sobel2,cmap = 'gray'),plt.title('Sobel Edge
Detection'), plt.axis('off')
plt.subplot2grid((5,2),(4,1)),plt.imshow(thresh2,cmap = 'gray'),plt.title('After Inverse
Thresholding'), plt.axis('off')
plt.savefig("E:/unnati/DIP/download1.jpeg", format="jpeg",dpi=1200)
plt.show()
```

INPUT:

OUTPUT:



PROGRAM: COMPARISON OF LAPLACIAN, SOBEL EDGE DETECTION

```
import cv2
import matplotlib.pyplot as plt
# load the image and convert to grayscale
inpimage = cv2.imread("E:/unnati/DIP/download.jpeg")
inpimage = cv2.cvtColor(inpimage, cv2.COLOR_BGR2RGB)
gray = cv2.cvtColor(inpimage, cv2.COLOR_BGR2GRAY)
#Denoising image before edge detection
denoiseimg = cv2.GaussianBlur(gray,(3,3),0)
#Apply Sobel operator on x and y dimensions
sobelx = cv2.Sobel(denoiseimg,cv2.CV_64F,1,0,ksize=5) # x
sobely = cv2.Sobel(denoiseimg,cv2.CV_64F,0,1,ksize=5) # y
sobel = sobelx + sobely
laplacian = cv2.Laplacian(denoiseimg,cv2.CV_64F)
#Apply Inverse Thresholding to Sobel Edge Detection
ret,thresh1 = cv2.threshold(sobel,220,255, cv2.THRESH_BINARY_INV)
#Apply Inverse Thresholding to Laplacian Edge Detection
ret,thresh2 = cv2.threshold(laplacian,10,255, cv2.THRESH_BINARY_INV)
plt.subplot(321),plt.imshow(inpimage),plt.title('Original Image'), plt.axis('off')
plt.subplot(322),plt.imshow(gray,cmap='gray'),plt.title('Grayscale Image'), plt.axis('off')
plt.subplot(323),plt.imshow(sobel,cmap='gray'),plt.title('Sobel Edge Detection'), plt.axis('off')
plt.subplot(324),plt.imshow(laplacian,cmap='gray'),plt.title('Laplacian Edge Detection'),
plt.axis('off')
plt.subplot(325),plt.imshow(thresh1,cmap='gray'),plt.title('Sobel after thresholding'),
plt.axis('off')
plt.subplot(326),plt.imshow(thresh2,cmap='gray'),plt.title('Laplacian after Thresholding'),
plt.axis('off')
plt.savefig("D:/Diana/DIPLAB/sobellap.png", format="png",dpi=1200)
plt.show()
```

OUTPUT:

Original Image

Grayscale Image

Sobel Edge Detection

Laplacian Edge Detection

Sobel after thresholding

Laplacian after Thresholding

ADDITIONAL PROGRAM

PROGRAM 1: Perform Image Transformation using OpenCV

1. Translation

2. Scaling

3. Reflection

4. Rotation

5. Crop

6. Shearing

Image transformation is a coordinate changing function, it maps some (x, y) points in one coordinate system to points (x', y') in another coordinate system.

A] IMAGE TRANSLATION

Image translation is the rectilinear shift of an image from one location to another, so the shifting of an object is called translation. The matrix shown below is used for the translation of the image:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & B_x \\ 0 & 1 & B_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translate

The value of b_x defines how much the image will be moved on the x-axis and the value of b_y determines the movement of the image on the y-axis:

```
import numpy as np

import cv2

import matplotlib.pyplot as plt


# read the input image

img = cv2.imread("E:/unnati/DIP/house.jpeg")

# convert from BGR to RGB so we can plot using matplotlib

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# disable x & y axis

plt.axis('off')
```

```
# show the image
plt.imshow(img)
plt.show()
# get the image shape
rows, cols, dim = img.shape
# transformation matrix for translation
M = np.float32([[1, 0, 50],
                [0, 1, 50],
                [0, 0, 1]])
# apply a perspective transformation to the image
translated_img = cv2.warpPerspective(img, M, (cols, rows))
# disable x & y axis
plt.axis('off')
# show the resulting image
plt.imshow(translated_img)
plt.show()
# save the resulting image to disk
plt.imsave("city_translated.jpg", translated_img)
```

INPUT :

OUTPUT:

B] IMAGE SCALING

Image scaling is a process used to resize a digital image. OpenCV has a built-in function cv2.resize(), but we will perform transformation using matrix multiplication as previously. The matrix used for scaling is shown below:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scale

```
import numpy as np

import cv2

import matplotlib.pyplot as plt


# read the input image

img = cv2.imread("E:/unnati/DIP/house.jpeg")

# convert from BGR to RGB so we can plot using matplotlib

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# disable x & y axis

plt.axis('off')

# show the image

plt.imshow(img)

plt.show()

# get the image shape

rows, cols, dim = img.shape

#transformation matrix for Scaling
```

```
M = np.float32([[1.5, 0  , 0],
                [0,   1.8, 0],
                [0,   0,   1]])
# apply a perspective transformation to the image
scaled_img = cv2.warpPerspective(img,M,(cols*2,rows*2))
# disable x & y axis
plt.axis('off')
# show the resulting image
plt.imshow(scaled_img)
plt.show()
# save the resulting image to disk
plt.imsave("city_scaled.jpg", scaled_img)
```

INPUT:

OUTPUT:

C] IMAGE REFLECTION

Image reflection (or mirroring) is useful for flipping an image, it can flip the image vertically as well as horizontally, which is a particular case of scaling. For reflection along the x-axis, we set the value of Sy to -1, and Sx to 1, and vice-versa for the y-axis reflection.

The transformation matrix for reflection is shown below:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & \text{rows} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Reflection x-axis

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & \text{cols} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Reflection y-axis

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

# read the input image
img = cv2.imread("E:/unnati/DIP/house.jpeg")
# convert from BGR to RGB so we can plot using matplotlib
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# disable x & y axis
plt.axis('off')
# show the image
plt.imshow(img)
plt.show()
# get the image shape
rows, cols, dim = img.shape
# transformation matrix for x-axis reflection
M = np.float32([[1,  0, 0   ],
                [0, -1, rows],
                [0,  0, 1   ]])
```

```
# transformation matrix for y-axis reflection
# M = np.float32([[-1, 0, cols],
#                 [ 0, 1, 0   ],
#                 [ 0, 0, 1   ]])
# apply a perspective transformation to the image
reflected_img = cv2.warpPerspective(img,M,(int(cols),int(rows)))
# disable x & y axis
plt.axis('off')
# show the resulting image
plt.imshow(reflected_img)
plt.show()
# save the resulting image to disk
plt.imsave("city_reflected.jpg", reflected_img)
```

INPUT:

OUTPUT:

X axis

Y axis

D] IMAGE ROTATION

Rotation is a concept in mathematics that is a motion of a certain space that preserves at least one point. Image rotation is a common image processing routine with applications in matching, alignment, and other image-based algorithms, it is also extensively in data augmentation, especially when it comes to image classification.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2D *in-plane* rotation

```
import numpy as np

import cv2

import matplotlib.pyplot as plt


# read the input image

img = cv2.imread("E:/unnati/DIP/house.jpeg")

# convert from BGR to RGB so we can plot using matplotlib

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# disable x & y axis

plt.axis('off')

# show the image

plt.imshow(img)

plt.show()

# get the image shape

rows, cols, dim = img.shape

#angle from degree to radian
```

```
angle = np.radians(10)
#transformation matrix for Rotation
M = np.float32([[np.cos(angle), -(np.sin(angle)), 0],
                [np.sin(angle), np.cos(angle), 0],
                [0, 0, 1]])
# apply a perspective transformation to the image
rotated_img = cv2.warpPerspective(img, M, (int(cols),int(rows)))
# disable x & y axis
plt.axis('off')
# show the resulting image
plt.imshow(rotated_img)
plt.show()
# save the resulting image to disk
plt.imsave("city_rotated.jpg", rotated_img)
```

INPUT:

OUTPUT

E] IMAGE CROPPING

Image cropping is the removal of unwanted outer areas from an image

```
import numpy as np
```

```
import cv2
import matplotlib.pyplot as plt

# read the input image
img = cv2.imread("E:/unnati/DIP/house.jpeg")
# convert from BGR to RGB so we can plot using matplotlib
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# disable x & y axis
plt.axis('off')
# show the image
plt.imshow(img)
plt.show()
# get 200 pixels from 100 to 300 on both x-axis & y-axis
# change that if you will, just make sure you don't exceed cols & rows
cropped_img = img[100:300, 100:300]
# disable x & y axis
plt.axis('off')
# show the resulting image
plt.imshow(cropped_img)
plt.show()
# save the resulting image to disk
plt.imsave("city_cropped.jpg", cropped_img)
```

INPUT:

OUTPUT:

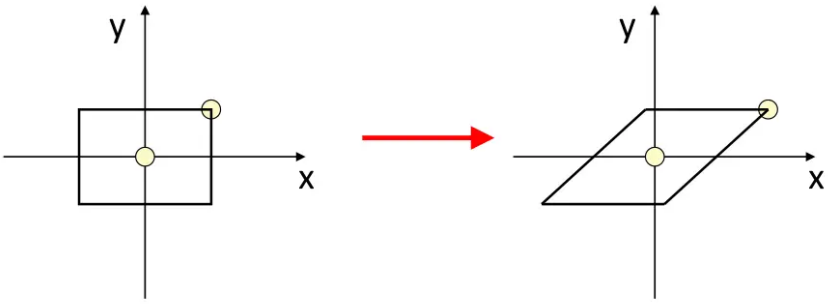
F] IMAGE SHEARING

Shear mapping is a linear map that displaces each point in a fixed direction, it substitutes every point horizontally or vertically by a specific value in proportion to its x or y coordinates, there are two types of shearing effects.

Shearing in the x-axis Direction

When shearing is done in the x-axis direction, the boundaries of the image that are parallel to the x-axis keep their location, and the edges parallel to the y-axis change their place depending on the shearing factor:

Image Shearing in the x-axis direction



Shearing in the y-axis Direction

When shearing is done in the y-axis direction, the boundaries of the image that are parallel to the y-axis keep their location, and the edges parallel to the x-axis change their place depending on the shearing factor.

The matrix for shearing is shown in the below figure:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Shear

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

# read the input image
img = cv2.imread("E:/unnati/DIP/house.jpeg")
# convert from BGR to RGB so we can plot using matplotlib
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# disable x & y axis
plt.axis('off')
# show the image
plt.imshow(img)
plt.show()
# get the image shape
rows, cols, dim = img.shape
# transformation matrix for Shearing
# shearing applied to x-axis
M = np.float32([[1, 0.5, 0],
                [0, 1  , 0],
                [0, 0  , 1]])
# shearing applied to y-axis
# M = np.float32([[1,   0, 0],
#                 [0.5, 1, 0],
#                 [0,   0, 1]])
# apply a perspective transformation to the image
sheared_img = cv2.warpPerspective(img,M,(int(cols*1.5),int(rows*1.5)))
# disable x & y axis
plt.axis('off')
# show the resulting image
plt.imshow(sheared_img)
```

```
plt.show()
# save the resulting image to disk
plt.imsave("city_sheared.jpg", sheared_img)
```

INPUT:

OUTPUT:

X Direction Shear

Y Direction Shear

PROGRAM 2: Implement Lossy and Lossless Compression Technique using OpenCV

Lossy reduces file size by permanently removing some of the original data. Lossless reduces file size by removing unnecessary metadata.

```
import cv2

import numpy as np

import os


img = cv2.imread("E:/unnati/DIP/house.jpeg")

#Save the Image with Lossless Compression (PNG)

cv2.imwrite('lossless_compressed_image.png', img)

#Save the Image with Lossy Compression (JPEG)

jpeg_quality = 90  # A value between 0 and 100 (higher means better quality, but larger file size)

cv2.imwrite('lossy_compressed_image.jpg', img, [cv2.IMWRITE_JPEG_QUALITY, jpeg_quality])

#Compare the Compression Results we are importing os


original_size = os.path.getsize('E:/unnati/DIP/house.jpeg')

lossless_size = os.path.getsize('lossless_compressed_image.png')

lossy_size = os.path.getsize('lossy_compressed_image.jpg')


print(f'Original image size: {original_size} bytes')

print(f'Lossless compressed image size: {lossless_size} bytes')

print(f'Lossy compressed image size: {lossy_size} bytes')

#display the images

lossless_img = cv2.imread('lossless_compressed_image.png')

lossy_img = cv2.imread('lossy_compressed_image.jpg')

cv2.imshow('Original image', img)

cv2.imshow('Lossless image', lossless_img)

cv2.imshow('Lossy image', lossy_img)

cv2.waitKey(0)
```

cv2.destroyAllWindows()

INPUT:

OUTPUT: