

**Vidyavardhini’s**

**College of Engineering & Technology**

Vasai Road (W)

**Department of**

**Electronics and Telecommunication Engineering**

**Lab Manual**

|  |  |  |  |
| --- | --- | --- | --- |
| Semester | VI | Class | TE |
| Course Code | ECL603 | Academic Year | Rev-2019 Scheme |
| Course Name | Image Processing and Machine Vision Lab | | |
| Name of Faculty | Mrs. Trupti Shah | | |
| Supporting Staff | Mrs. Madhu Lade | | |

****

**Vidyavardhini’s College of Engineering & Technology**

**Vision**

To be a premier institution of technical education, aiming at becoming a valuable resource for industry and society.

**Mission**

* To provide technologically inspiring environment for learning.
* To promote creativity, innovation and professional activities.
* To inculcate ethical and moral values.
* To cater personal, professional and societal needs through quality education.

**Department Vision:**

To contrive educational and research environment to serve industry and society needs in the field of electronics and telecommunication engineering.

**Department Mission:**

1. To enrich soft skills, ethical values, environmental and societal awareness.
2. To develop technical proficiency through projects and laboratory work.
3. To encourage students for lifelong learning through interaction with outside world.

**Program Education Objectives (PEOs):**

* The graduates will exhibit knowledge of mathematics, science, electronics, and communication, and will be able to apply the same in diversified field.
* The graduates will develop a habit of continuous learning while working in multidisciplinary environment.
* The graduates will grow as an individual with proficiency in technical skills, ethical values, communication skills, teamwork and professionalism.

**Program Specific Outcomes (PSOs):**

At the end of the program engineering graduate will be able to:

1. Apply the knowledge of Electronics and Communication to analyse, design and implement application specific problems with modern tools.
2. Adapt emerging technologies with continuous learning in the field of Electronics and Telecommunication engineering with appropriate solutions to real life problems.

**Program Outcomes (POs):**

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 EXPERIMENT NOs, 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 modeling 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 teamwork:** 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.

**Sr. No. Content**

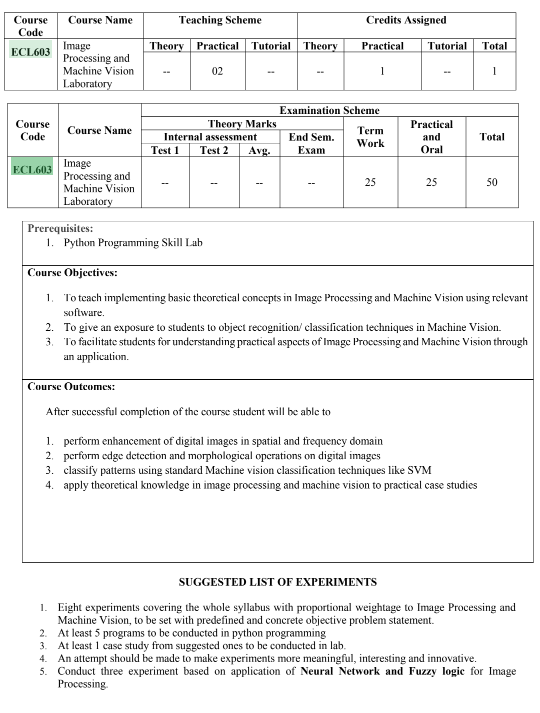
1. Syllabus

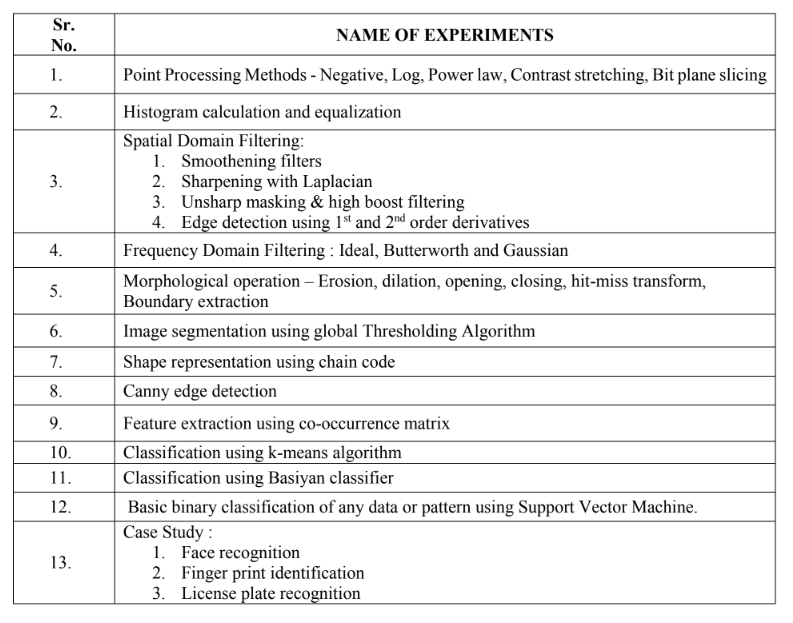
2. Course Objectives and Course Outcomes

3. Mapping of EXPERIMENT NOs with Course Outcomes

4. Mapping of COs with POs and PSOs

5. List of EXPERIMENT NOs

**Syllabus:**

****

**Course Objectives**

|  |  |
| --- | --- |
| 1 | To teach implementing basic theoretical concepts in Image Processing and Machine Vision using relevant software. |
| 2 | To give an exposure to students to object recognition/ classification techniques in Machine Vision. |
| 3 | To facilitate students for understanding practical aspects of Image Processing and Machine Vision through an application. |

**Course Outcomes**

|  |  |  |  |
| --- | --- | --- | --- |
| At the end of the course student will be able to: | | Action Verb | Bloom’s Level |
| ECL603.1 | Analyse grey scale resolution using point processing algorithms in python. | Analyse | 4 |
| ECL603.2 | Implement histogram equalisation for image enhancement using Python | Implement | 3 |
| ECL603.3 | Analyse spatial domain and frequency domain filtering for image enhancement using python. | Analyse | 3 |
| ECL603.4 | Apply morphological operations for various image processing applications using python. | Apply | 3 |
| ECL603.5 | Apply basic concepts of Neural Networks for image processing using python. | Apply | 3 |
| ECL603.6 | Apply image processing base algorithms for real time applications. | Apply | 3 |

**List of EXPERIMENT NOs**

|  |  |
| --- | --- |
| Sr. No | Name of EXPERIMENT NOs |
| 1 | Write a program in python to perform Point Processing techniques on digital image to obtain:   * Negative image. * Gamma transformation. |
| 2 | Write a program in python to analyse contribution of each bit by bit plane slicing technique. |
| 3 | Write a program in python to perform image enhancement using histogram equalization. |
| 4 | Write a program in python to perform spatial domain filtering for smoothening and sharpening the filter. |
| 5 | Write a program in python to perform frequency domain filtering:   * Apply FFT on given image * Perform low pass and high pass filtering in frequency domain. * Apply IFFT to reconstruct image. |
| 6 | Write a program in python for edge detection using canny edge detector. |
| 7 | Write a program in python to perform following morphological operations   * Erosion, dilation, * Opening and Closing, * Hit-miss transforms. |
| 8 | Write a program in python to find Chain code for 2-D Line. |
| 9 | Write a program in python to apply Support Vector Machine to split data set. |
| 10 | Write a program in python to perform pattern clustering using K-means algorithm. |
| 11 | Write a program in python to apply MP Model to train network. |
| 12 | Write a program in python to perform CNN using VGG16 model. |
| 13 | Case Study |

**EXPERIMENT NO – 1**

**Aim:** Write a program in python to perform Point Processing techniques on digital image to obtain

* Negative image.
* Gamma transformation.

**Software:** Python

**Theory:**

Point processing is now defined as an operation which calculates the new value of pixel in

g(x,y) based on the value of the pixel in the same position in f(x,y) and some operation. That

is, the values of a pixel's neighbors in f(x,y) have no effect whatsoever, hence the name point

processing.

Following are some of the common examples of point processing

1. Digital negative
2. Contrast stretching
3. Power law
4. Log Transformation
5. Bit plane slicing

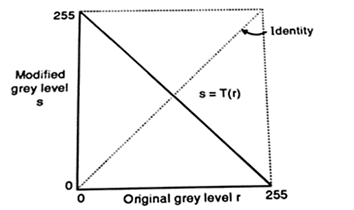
**Digital Negative:**  Digital negative, simply means inverting the grey levels, which means black in the original image will be white in digital negative and vice versa. The figure below is the digital negative transformation for an 8-bit image. The digital negative can be obtained by a simple transformation given by s=255–r (Where rmax=255)

Hence when r=0, s=255 and when r=255 ,s=0

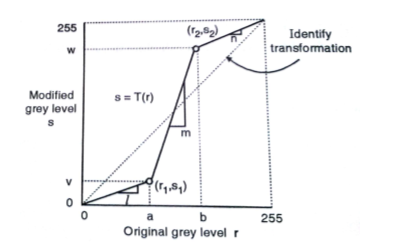
In general,

s=(L−1)−r

Where L = Number of grey levels



**Contrast stretching: -** Contrast stretching, in general increases the contrast of the images, by making the dark portions darker and the bright portion brighter. Figure below shows the transformation used to achieve contrast stretching



The formulation for contrast stretching algorithm is given below,

s=l.r    0<=r<=a

s=m.(r−a)+v    a<=r<=b

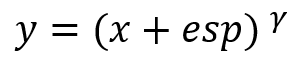
s=n.(r−b)+w    b<=r<=(L−1)

In the above equation, l, m and n are slopes, and it is clear from the figure that l and m are less than 1 whereas m is greater than 1, this increases the dynamic range of the image.

**Power Law Transformation:-**

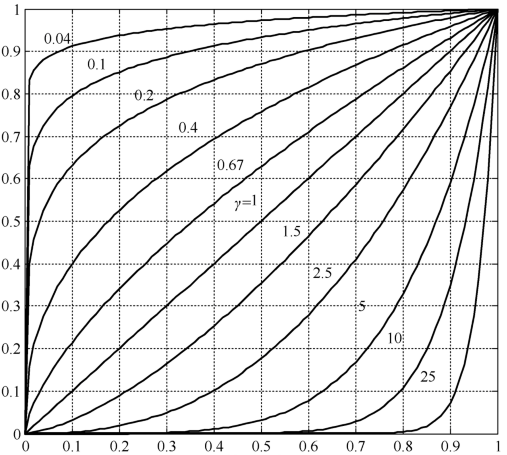
The gamma transform, also known as exponential transformation or power transformation, is a commonly used grayscale non-linear transformation.

The mathematical expression of the gamma transformation is as follows:



Where x and y are the intensity level of a pixel in input and output, “esp” is the compensation coefficient, and γ is the gamma coefficient. When performing the conversion, it is a common practice to first convert the intensity level from the range of 0 to 255 to 0 to 1. Then perform the gamma conversion and at last restore to the original range.

When setting esp = 1, we can get a mapping form like the following, with the gray intensity of the input in the horizontal axis and the gray intensity of the output on the vertical axis.



Plots of the equation [formula] for various values of γ (c = 1 in all cases).

All curves were scaled to fit in the range shown. x-axis represents Input intensity level, r and y-axis represents the output Intensity Level S

The gamma transformation can selectively enhance the contrast of the dark region or the light region depending on the value of γ.

* When γ > 1, the contrast of the light grey area is enhanced. Take γ = 25 for example, the pixels with the range of 0.8-1 (at the scale of 256, it corresponds to 240-255) are mapped to the range of 0-1
* When γ < 1, the contrast of the dark grey area is enhanced
* When γ = 1, this transformation is linear, that is, the original image is not changed

**Log Transformation: -**

Log transformation of an image means replacing all pixel values, present in the image, with its logarithmic values. Log transformation is used for image enhancement as it expands dark pixels of the image as compared to higher pixel values.

The Log transformation can be defined by the formula:

s = c log(r+1)

where s and r are the pixel values of the output and the input image and c is a constant.

1 is added to make the minimum value at least 1.

**Result Analysis and Conclusion**:

**Post Experiment Questions:**

* How does inverting an image using the negative transformation affect visual perception
* Investigate the impact of different gamma correction values on the perceived brightness

**Code:**

**Negation of image:**

import cv2

import matplotlib.pyplot as plt

url="/home/kimaya/Documents/Engineering Third Year/Engineering semVI/IPMV pracs/camera man.jpg"

img=cv2.imread(url)

# plt.imshow(img)

# plt.show()

height,width,\_=img.shape

for i in range(0, height - 1):

for j in range(0, width - 1):

# Get the pixel value

pixel = img[i, j]

# Negate each channel by

# subtracting it from 255

# 1st index contains red pixel

pixel[0] = 255 - pixel[0]

# 2nd index contains green pixel

pixel[1] = 255 - pixel[1]

# 3rd index contains blue pixel

pixel[2] = 255 - pixel[2]

# Store new values in the pixel

img[i, j] = pixel

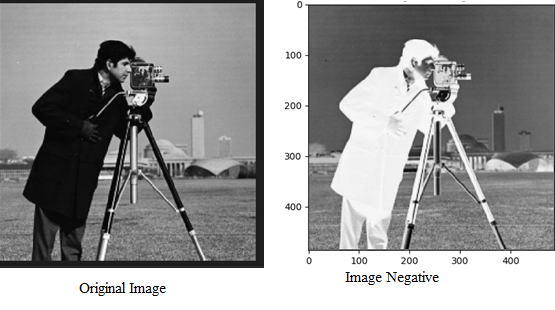
# Display the negative transformed image

plt.imshow(img)

plt.title("negative image")

plt.show()

**Output:**



**Gamma Transformation:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Function to map each intensity level to output intensity level.

def pixelVal(pix, r1, s1, r2, s2):

if (0 <= pix and pix <= r1):

return (s1 / r1)\*pix

elif (r1 < pix and pix <= r2):

return ((s2 - s1)/(r2 - r1)) \* (pix - r1) + s1

else:

return ((255 - s2)/(255 - r2)) \* (pix - r2) + s2

# Open the image.

url="/home/kimaya/Documents/Engineering Third Year/Engineering semVI/IPMV pracs/camera man.jpg"

img = cv2.imread(url)

# Define parameters.

r1 = 70

s1 = 1

r2 = 140

s2 = 255

# Vectorize the function to apply it to each value in the Numpy array.

pixelVal\_vec = np.vectorize(pixelVal)

# Apply contrast stretching.

contrast\_stretched = pixelVal\_vec(img, r1, s1, r2, s2)

# Save edited image.

cv2.imwrite('contrast\_stretch.jpg', contrast\_stretched)

plt.imshow(contrast\_stretched)

plt.show()

# Trying 4 gamma values.

for gamma in [0.1, 0.5, 1.2, 2.2]:

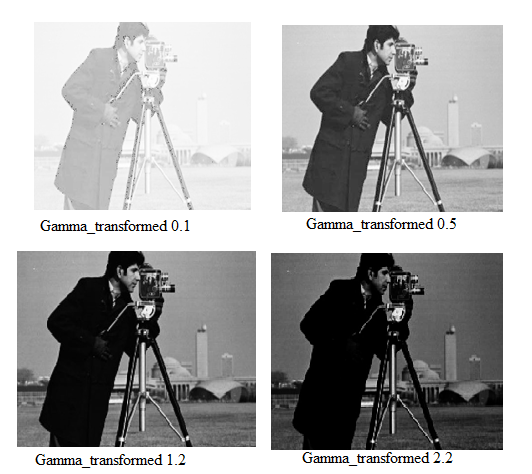
# Apply gamma correction.

gamma\_corrected = np.array(255\*(img / 255) \*\* gamma, dtype = 'uint8')

cv2.imwrite('gamma\_transformed'+str(gamma)+'.jpg', gamma\_corrected)

plt.imshow(gamma\_corrected)

plt.show()



**EXPERIMENT NO - 2**

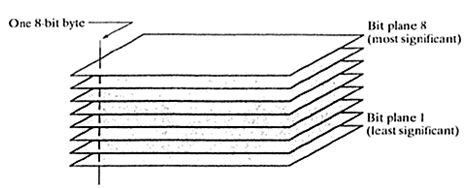
**Aim:** Write a program in python to analyse contribution of each bit-by-bit plane slicing technique.

**Software:** Python.

**Theory:** In this technique, we find the contribution made by each bit in the final image.

Pixels are digital numbers composed of bits, for example the intensity of each pixel in 256 level grey scale image is composed of 8-bits.

Instead of highlighting intensity level ranges, we could highlight the contribution made to the total image by specific bits. Figure below shows how an 8-bit image may be considered as being composed of eight one bit planes, showing the significance of each bit.



**Fig.: Bit Planes for 8 Bit Image**

In bit plane slicing, we will get 8 different images and all 8 images will be binary, showing the significance of each bit in the 8-bit image.

Decomposing an image into its bit planes is useful for analyzing the creative importance of each bit in the image, a process that aids in determining the adequacy of the number of bits used to quantize the image.

Also, this type of decomposition is useful for image compression in which, fewer than all planes are used in reconstructing the image.

**Task:**

1. Read the image.

2 Perform the various point processing methods on the image.

3. Display the original image and results.

**Result Analysis and Conclusion**:

**Post Experiment Questions:**

* How does the visual quality and information retention of an image change when different bit planes (e.g., most significant bits vs. least significant bits) are sliced.
* 3 bit image is given below draw bit planes of the image.

|  |  |  |
| --- | --- | --- |
| **5** | **3** | **2** |
| **6** | **3** | **4** |
| **1** | **5** | **7** |

**Code:**

#Bit plane slicing

import numpy as np

import cv2

# Read the image in greyscale

wheel = r"/home/kimaya/Documents/Engineering Third Year/Engineering semVI/IPMV pracs/Bit\_plane\_testimg.jpeg"

img = cv2.imread(wheel,0)

#Iterate over each pixel and change pixel value to binary using np.binary\_repr() and store it in a list.

lst = []

for i in range(img.shape[0]):

for j in range(img.shape[1]):

lst.append(np.binary\_repr(img[i][j] ,width=8)) # width = no. of bits

# We have a list of strings where each string represents binary pixel value. To extract bit planes we need to iterate over the strings and store the characters corresponding to bit planes into lists.

# Multiply with 2^(n-1) and reshape to reconstruct the bit image.

eight\_bit\_img = (np.array([int(i[0]) for i in lst],dtype = np.uint8) \* 128).reshape(img.shape[0],img.shape[1])

seven\_bit\_img = (np.array([int(i[1]) for i in lst],dtype = np.uint8) \* 64).reshape(img.shape[0],img.shape[1])

six\_bit\_img = (np.array([int(i[2]) for i in lst],dtype = np.uint8) \* 32).reshape(img.shape[0],img.shape[1])

five\_bit\_img = (np.array([int(i[3]) for i in lst],dtype = np.uint8) \* 16).reshape(img.shape[0],img.shape[1])

four\_bit\_img = (np.array([int(i[4]) for i in lst],dtype = np.uint8) \* 8).reshape(img.shape[0],img.shape[1])

three\_bit\_img = (np.array([int(i[5]) for i in lst],dtype = np.uint8) \* 4).reshape(img.shape[0],img.shape[1])

two\_bit\_img = (np.array([int(i[6]) for i in lst],dtype = np.uint8) \* 2).reshape(img.shape[0],img.shape[1])

one\_bit\_img = (np.array([int(i[7]) for i in lst],dtype = np.uint8) \* 1).reshape(img.shape[0],img.shape[1])

#Concatenate these images for ease of display using cv2.hconcat()

finalr = cv2.hconcat([eight\_bit\_img,seven\_bit\_img,six\_bit\_img,five\_bit\_img])

finalv =cv2.hconcat([four\_bit\_img,three\_bit\_img,two\_bit\_img,one\_bit\_img])

# Vertically concatenate

final = cv2.vconcat([finalr,finalv])

# Display the images

cv2.imshow('a',final)

cv2.waitKey(0)

**Output:**

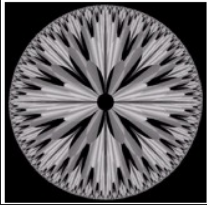


Fig. Original image:

Bit Plane Slicing:

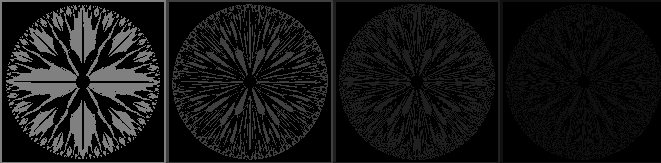


Fig. Bit planes of input image

**EXPERIMENT NO - 3**

**Aim:** Write a program in python to perform image enhancement using histogram equalization.

**Software:** Python.

**Theory:** Histogram of a digital image with gray levels in the range [0, L-1] is a discrete function

*h*(*rk*) *= nk*

where *rk* = *k*th gray level

*nk* = no. of pixels in the image having gray level *rk* .

Histograms are frequently normalized by the total number of pixels in the image. Assuming a Shape

Description automatically generated with medium confidence image, a normalized histogram

Shape

Description automatically generated with medium confidence

is related to probability of occurrence of *rk* in the image. The histogram conveys the basic understanding of the appearance of the image as shown in Fig. 1.

Chart, diagram

Description automatically generated

Fig. 1

**Histogram Equalization** :

Histogram Equalization is a form of Automatic Contrast Stretching. Here the contrast stretching is done on the entire image automatically without requiring any parameter selection It transforms *r* to *s* through *T*(*r*) such that the histogram of output image is flat.

Consider *r* = [0,1] be normalized gray levels. The Transformation function is

*s* = *T*(*r*) 0 ≤ *r* ≤ 1

*T*(*r*) must satisfy two conditions:

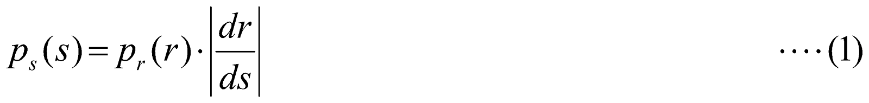
It should be single valued & monotonically increasing function

*0 ≤ T(r) ≤ 1 0 ≤ r ≤ 1*

If the above conditions are satisfied then we can write the inverse transformation function from *s* to *r* as

*r = T –1(s) 0 ≤ s ≤ 1*

Consider *pr*(*r*) and *ps*(s) be the PDFs of input and output gray levels. If *pr*(*r*) and *T*(*r*) is known and *T*-1(*s*) exists then



Consider the transformation function

Shape

Description automatically generated with medium confidence

where *w* = dummy variable for integration.

Here RHS is CDF (Cumulative Distribution Function) of random variable *r*. Differentiating eq. (2) w.r.t. *r*

Shape

Description automatically generated with medium confidence

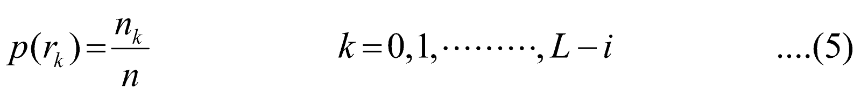
Substituting in eq. (1)

Shape

Description automatically generated with medium confidence

which is nothing but a uniform distribution.

For digital images, the equations are modified as



and

Shape

Description automatically generated with medium confidence

**Task :**

1. Read the original image.

2. Calculate the histogram of the original image.

3. Perform contrast stretching using histogram equalization.

4. Calculate the histogram of the equalized image.

5. Display the original image, equalized image, the input histogram, the output histogram and the transformation function.

**Result Analysis and Conclusion**:

**Post Experiment Questions:**

* What is histogram equalization in the context of image processing, and how does it contribute to enhancing the visual quality of an image?
* Choose a sample grayscale image and describe step-by-step how histogram equalization works to improve the contrast and details in the image.

**Code:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('mona lisa.jpg',0)

hist,bins = np.histogram(img.flatten(),256,[0,256])

cdf = hist.cumsum()

cdf\_normalized = cdf \* hist.max()/ cdf.max()

plt.plot(cdf\_normalized, color = 'b')

plt.hist(img.flatten(),256,[0,256], color = 'r')

plt.xlim([0,256])

plt.legend(('cdf','histogram'), loc = 'upper left')

plt.show()

cdf\_m = np.ma.masked\_equal(cdf,0)

cdf\_m = (cdf\_m - cdf\_m.min())\*255/(cdf\_m.max()-cdf\_m.min())

cdf = np.ma.filled(cdf\_m,0).astype('uint8')

img2 = cdf[img]

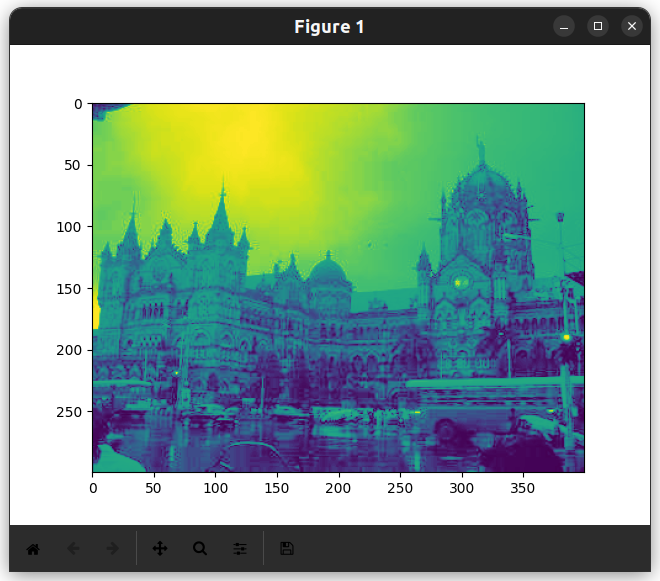
equ = cv2.equalizeHist(img)

res = np.hstack((img,equ)) #stacking images side-by-side

cv2.imwrite('res.png',res)

**Output:**

|  |  |
| --- | --- |
| Initial histogram | Equalized histogram |

****

Equalized Image

**EXPERIMENT NO-4**

**Aim:** Write a program in python to perform spatial domain filtering for smoothening and sharpening the filter.

**Software:** Python

**Theory:** Filtering is a technique for modifying or enhancing an image. Spatial domain operation or filtering (the processed value for the current pixel processed value for the current pixel depends on both itself and surrounding pixels). Hence Filtering is a neighborhood operation, in which the value of any given pixel in the output image is determined by applying some algorithm to the values of the pixels in the neighborhood of the corresponding input pixel. A pixel's neighborhood is some set of pixels, defined by their locations relative to that pixel.

**Smoothing Spatial Filter:** Smoothing filter is used for blurring and noise reduction in the image. Blurring is pre-processing steps for removal of small details and Noise Reduction is accomplished by blurring.

**Edge detection using 1st and 2nd order derivatives:**

First order derivative:

* Must be zero in flat segments.
* Must be non zero at the onset of a grey level step.
* Must be non zero along ramps.

First order derivative in 1-D is given by:

f' = f(x+1) - f(x)

Second order derivative:

* Must be zero in flat areas.
* Must be zero at the onset and end of a ramp.
* Must be zero along ramps.

Second order derivative in 1-D is given by:

f'' = f(x+1) + f(x-1) - 2f(x)

**Task :**  1. Read the image.

2. Perform different techniques of Spatial Domain Filtering.

3. Display the original image and results.

**Result Analysis and Conclusion**:

**Post Experiment Questions:**

* How does applying a smoothening filter affect the visual appearance of an image, and in what situations might it be beneficial to use such filters in image processing?
* What is high boost filtering in image processing, and how does it enhance image details?

**Code:**

**Image Blurring**

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread(' *Your image path* ')

blur = cv2.blur(img,(15,15))

plt.subplot(121),plt.imshow(img),plt.title('Original')

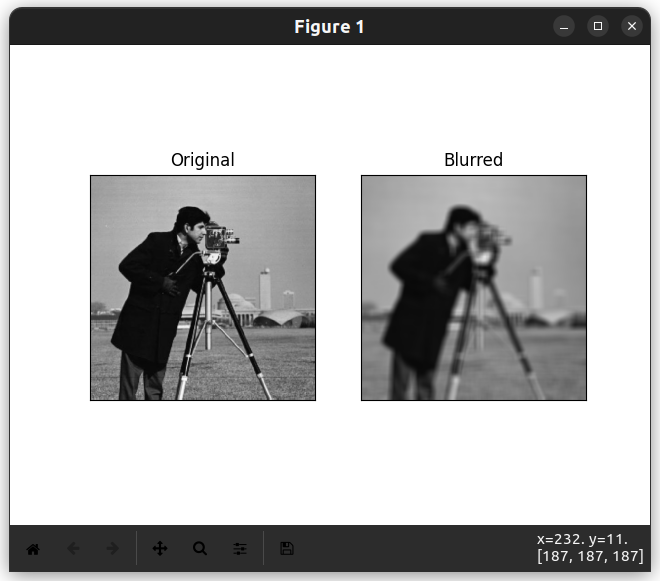
plt.xticks([]), plt.yticks([])

plt.subplot(122),plt.imshow(blur),plt.title('Blurred')

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

plt.show()

**Output:**

****

**Fig. Output of Low Pass Filter**

**Sobel and Laplacian filter**

#spatial domain filtering

import cv2

import numpy as np

from matplotlib import pyplot as plt

# loading image

img0 = cv2.imread('/home/kimaya/Documents/Engineering Third Year/Engineering semVI/IPMV pracs/sudoku.jpeg',)

# converting to gray scale

gay = 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')

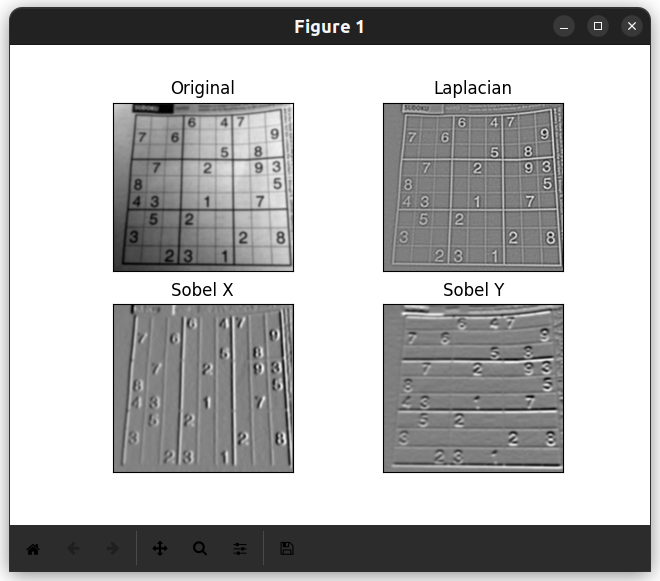
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')

plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])

plt.show()

**Output:**

****

**Fig. Output of Laplacian, Sobel Transform**

**EXPERIMENT NO– 5**

**Aim:** Write a program in python to perform frequency domain filtering:

* Apply FFT on given image
* Perform low pass and high pass filtering in frequency domain.
* Apply IFFT to reconstruct image.

**Software:** Python

**Theory:** Filtering is a technique for modifying or enhancing an image. Spatial domain operation or filtering (the processed value for the current pixel processed value for the current pixel depends on both itself and surrounding pixels). Hence Filtering is a neighborhood operation, in which the value of any given pixel in the output image is determined by applying some algorithm to the values of the pixels in the neighborhood of the corresponding input pixel. A pixel's neighborhood is some set of pixels, defined by their locations relative to that pixel.

**Unsharp masking & high boost filtering:**

when we use a blurred or unsharp image to create a mask than that technique is known as Unsharp Masking.

Thus, unsharp masking first produces a mask m(x,y) as

****

where, f(x,y) is the original image and fb(x,y) is the blurred version of the original image.

Then this mask is added back to the original image which results in enhancing the high-frequency components.

****

where k specifies what portion of the mask to be added. When k= 1 this is known as Unsharp masking. For k>1 we call this as high-boost filtering because we are boosting the high-frequency components by giving more weight to the masked (edge) image.

We can also write the above two equations into one as the weighted average of the original and the blurred image.

****

**Task :**  1. Read the image.

2. Perform different techniques of Spatial Domain Filtering.

3. Display the original image and results.

**Result Analysis and Conclusion**:

**Post Experiment Questions:**

* What is frequency domain filtering in image processing, and how does it differ from spatial domain filtering?
* Choose a common image enhancement task, such as noise reduction explain how frequency domain filtering can be advantageous in achieving the desired results compared to spatial domain filtering methods.

**Code:**

#High Pass filter FFT

import cv2 as cv

import numpy as np

from matplotlib import pyplot as plt

img = cv.imread(" *your image path* ",0)

f = np.fft.fft2(img)

magnitude\_spectrum1 = 20\*np.log(np.abs(f))

# plt. imshow(magnitude\_spectrum1, cmap = 'gray')

# plt.show()

fshift= np.fft.fftshift(f)

magnitude\_spectrum= 20\*np.log(np.abs(fshift))

rows, cols = img.shape

crow, ccol = rows//2 , cols//2

fshift[crow-30: crow+31, ccol-30:ccol+31] = 0

f\_ishift = np.fft.ifftshift (fshift)

img\_back = np.fft.ifft2(f\_ishift)

img\_back = np.real(img\_back)

mag\_LPF = magnitude\_spectrum1-magnitude\_spectrum

mag\_LPF = np.subtract(img,img\_back)

# plt. imshow(mag\_LPF, cmap = 'gray')

# plt.show()

plt.subplot(231),plt.imshow(img, cmap = "gray")

plt.title("Input Image"), plt.xticks([]), plt.yticks([])

plt. subplot (232),plt. imshow(magnitude\_spectrum, cmap = 'gray')

plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])

plt. subplot (233),plt.imshow(img, cmap = 'gray')

plt.title("Input Image"), plt.xticks([]), plt.yticks([])

plt.subplot (234),plt. imshow(img\_back, cmap = 'gray')

plt.title("Image after HPF"), plt.xticks([]), plt.yticks([])

plt.subplot (235),plt. imshow(magnitude\_spectrum1, cmap = 'gray')

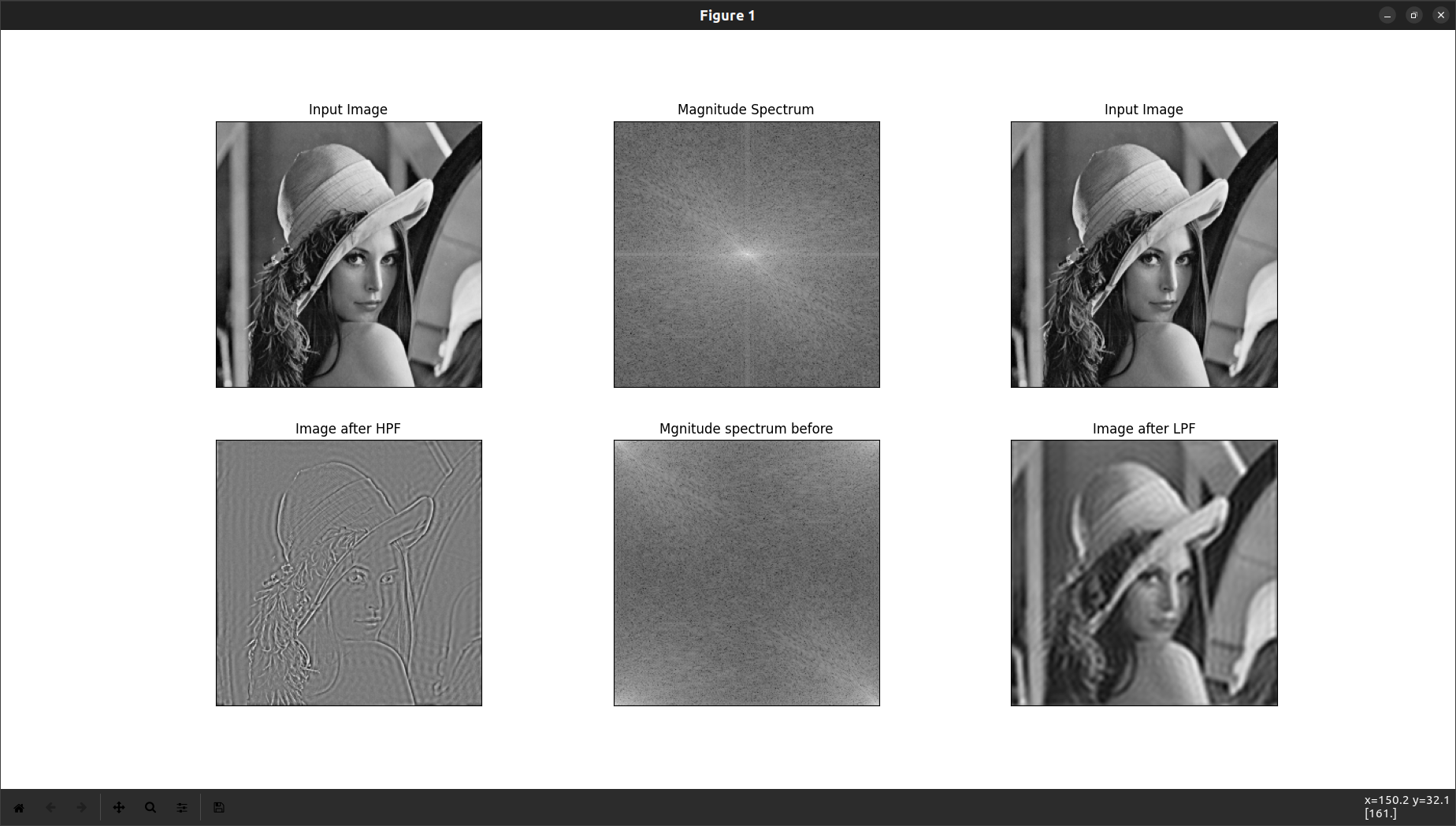
plt.title("Mgnitude spectrum before"), plt.xticks([]), plt.yticks([])

plt.subplot(236),plt. imshow(mag\_LPF, cmap = 'gray')

plt.title("Image after LPF"), plt.xticks([]), plt.yticks([])

plt.show()

**Output:**

****

**Fig.: Output of Frequency Domain Filter**

**EXPERIMENT NO– 6**

**Aim:** Write a program in python for edge detection using canny edge detector.

**Software:** Python.

**Theory:** The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images.

The Canny filter is a multi-stage edge detector. It uses a filter based on the derivative of a Gaussian in order to compute the intensity of the gradients. The Gaussian reduces the effect of noise present in the image. Then, potential edges are thinned down to 1-pixel curves by removing non-maximum pixels of the gradient magnitude. Finally, edge pixels are kept or removed using hysteresis thresholding on the gradient magnitude.

The Canny has three adjustable parameters: the width of the Gaussian (the noisier the image, the greater the width), and the low and high threshold for the hysteresis thresholding.

The general criteria for edge detection include:

1. Detection of edge with low error rate, which means that the detection should accurately catch as many edges shown in the image as possible
2. The edge point detected from the operator should accurately localize on the center of the edge.
3. A given edge in the image should only be marked once, and where possible, image noise should not create false edges.

**Task :**

1. Read the image.
2. Perform Canny Edge Detection.
3. Display the original image and results.

**Result Analysis and Conclusion**:

**Post Experiment Questions:**

* What is Canny edge detection, and what is its primary goal in image processing?
* Name and briefly explain the three main steps of Canny edge detection.

**Code:**

#canny edge detection

import cv2

img = cv2.imread(" *your image path* ") # Read image

# Setting parameter values

t\_lower = 50 # Lower Threshold

t\_upper = 150 # Upper threshold

# Applying the Canny Edge filter

edge = cv2.Canny(img, t\_lower, t\_upper)

cv2.imshow('original', img)

cv2.imshow('edge', edge)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Output:**

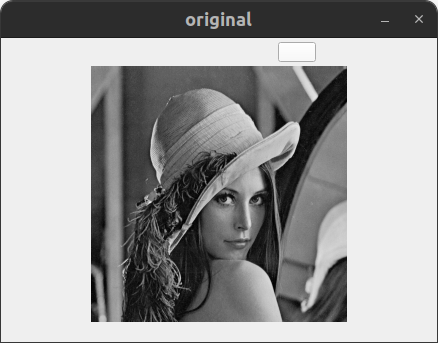
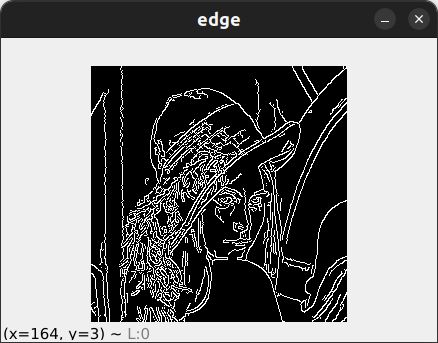
** **

Fig.: Input Image Fig. Canny Edge

**EXPERIMENT NO-7**

**Aim:** Write a program in python to perform following morphological operations

* Erosion and dilation
* Opening and Closing
* Hit-miss transforms.

**Software:** Python

**Theory:** Morphology is a broad set of image processing operations that process images based on shapes. Morphological operations apply a structuring element to an input image, creating an output image of the same size. In a morphological operation, the value of each pixel in the output image is based on a comparison of the corresponding pixel in the input image with its neighbors.

The most basic morphological operations are dilation and erosion.

**Dilation:** Dilation adds pixels to the boundaries of objects in an image. Dilation is A XOR B.

It increases the brightness of the objects.

**Erosion:** erosion removes pixels on object boundaries. Erosion is the dual of Dilation.

It removes the objects smaller than the structuring element.

The number of pixels added or removed from the objects in an image depends on the size and shape of the *structuring element* used to process the image. In the morphological dilation and erosion operations, the state of any given pixel in the output image is determined by applying a rule to the corresponding pixel and its neighbours in the input image. The rule used to process the pixels defines the operation as a dilation or an erosion.

**Opening**: Opening is generally used to restore or recover the original image to the maximum possible extent. Opening is a process in which first erosion operation is performed and then dilation operation is performed.

**Closing**: Closing is generally used to smoother the contour of the distorted image and fuse back the narrow breaks and long thin gulfs. Closing is also used for getting rid of the small holes of the obtained image. Closing is a process in which first dilation operation is performed and then erosion operation is performed

**Hit-miss transform:**

The Hit-or-Miss transformation is useful to find patterns in binary images. In particular, it finds those pixels whose neighbourhood matches the shape of a first structuring element B1 while not matching the shape of a second structuring element B2 at the same time. Mathematically, the operation applied to an image A can be expressed as follows:

A⊛B=(A⊖B1)∩(Ac⊖B2)

Therefore, the hit-or-miss operation comprises three steps:

* Erode image A with structuring element B1.
* Erode the complement of image A ( Ac) with structuring element B2.
* AND results from step 1 and step 2.

**Boundary extraction:**

The boundary of the image is different from the edges in the image. Edges represent the abrupt change in pixel intensity values while the boundary of the image is the contour. As the name boundary suggests that something whose ownership changes, in the image when pixel ownership changes from one surface to another, the boundary comes into the picture. Edge is basically the boundary line but the boundary is the line or location dividing the two surfaces.

**Task :**

1. Read the original image.
2. Define various Morphological operation and Execute them.
3. Display the original image and results.

**Result Analysis and Conclusion**:

**Post Experiment Questions:**

* Define dilation and erosion in the context of image morphology.
* What is a structuring element in image morphology, and how does its size and shape impact the outcome of morphological operations?

**Code:**

**Erosion & Dilation**

# dailation erosion

import cv2

import numpy as np

import matplotlib.pyplot as plt

img=cv2.imread("abc.jpeg")

kernel = np.ones((5,5), np.uint8)

img\_erosion = cv2.erode(img, kernel, iterations = 1)

img\_dilation = cv2.dilate(img, kernel, iterations = 1)

plt.title("Dilation")

plt.imshow(img\_dilation)

plt.show()

plt.title("Erosion")

plt.imshow(img\_erosion)

plt.show()

res=np.hstack((img\_dilation,img\_erosion))

plt.imshow(res)

plt.show()

**Output:**

|  |  |
| --- | --- |
| Fig. Input Image | Fig. Image Dilation |
| Fig. Image Erosion | |

**Opening:**

#opening

import cv2

import numpy as np

import matplotlib.pyplot as plt

img=cv2.imread('/home/kimaya/Documents/Engineering Third Year/Engineering semVI/IPMV pracs/ab.jpeg')

kernel=np.ones((5,5),np.uint8)

opening=cv2.morphologyEx(img, cv2.MORPH\_OPEN,kernel)

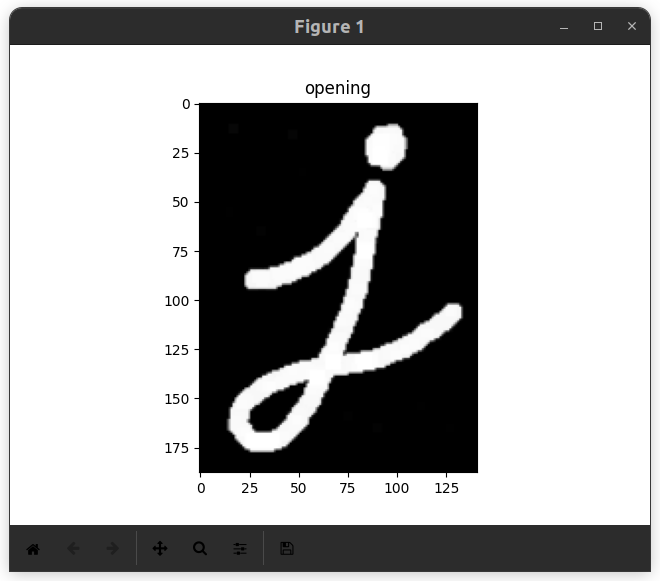
plt.title('Input')

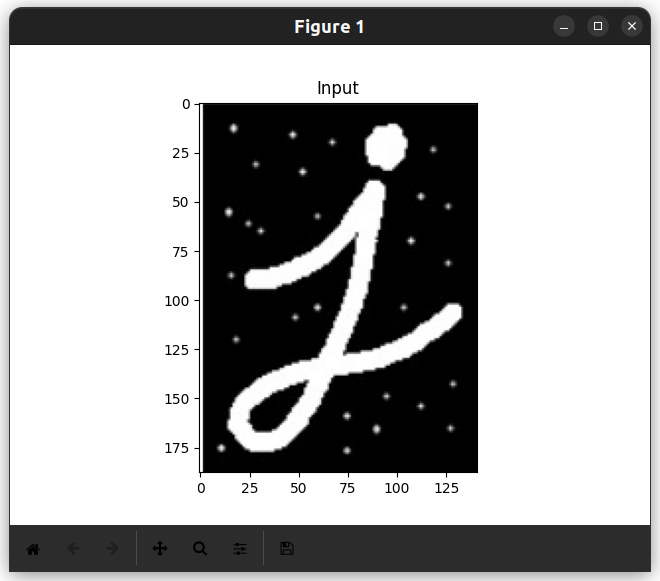
plt.imshow(img)

plt.show()

plt.title('opening')

plt.imshow(opening)

plt.show()

Fig. Input Image Fig. Opening Image

**Closing:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

img=cv2.imread('/home/kimaya/Documents/Engineering Third Year/Engineering semVI/IPMV pracs/closing.jpeg')

kernel=np.ones((5,5),np.uint8)

closing=cv2.morphologyEx(img, cv2.MORPH\_CLOSE,kernel)

plt.title('Input')

plt.imshow(img)

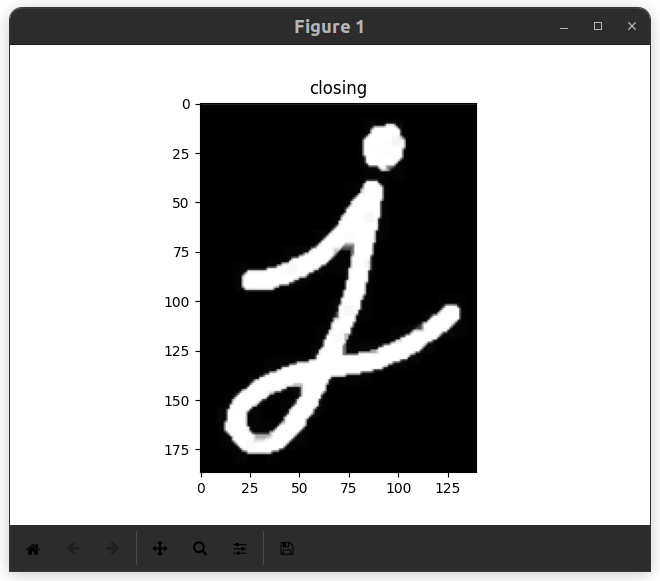
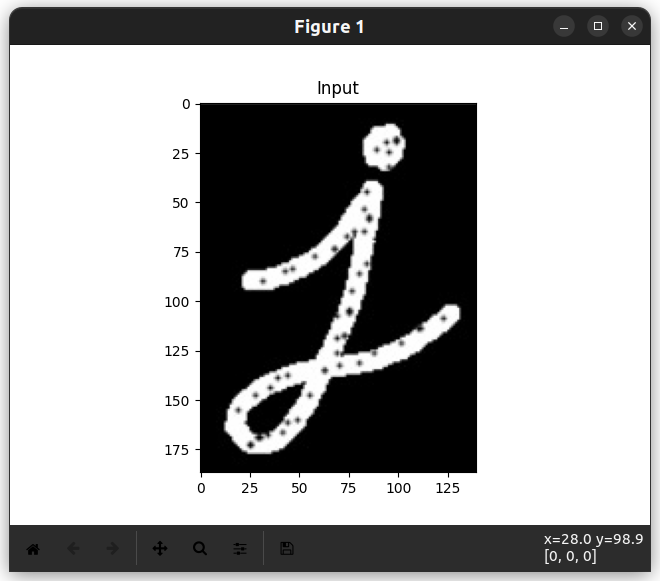
plt.show()

plt.title('closing')

plt.imshow(closing)

plt.show()

**Output:**

Fig. Input Image  Fig. Closed Image

**Hit-miss transform**

# HIT and MISS

import cv2 as cv

import numpy as np

import matplotlib.pyplot as plt

# generate image

input\_image = np.array((

[0, 0, 0, 0, 0, 0, 0, 0],

[0, 255, 255, 255, 0, 0, 0, 255],

[0, 255, 255, 255, 0, 0, 0, 0],

[0, 255, 255, 255, 0, 255, 0, 0],

[0, 0, 255, 0, 0, 0, 0, 0],

[0, 0, 255, 0, 0, 255, 255, 0],

[0,255, 0, 255, 0, 0, 255, 0],

[0, 255, 255, 255, 0, 0, 0, 0]), dtype="uint8")

kernel = np.array((

[0, 1, 0],

[1, -1, 1],

[0, 1, 0]), dtype="int")

output\_image = cv.morphologyEx(input\_image, cv.MORPH\_HITMISS, kernel)

rate = 50

kernel = (kernel + 1) \* 127

kernel = np.uint8(kernel)

kernel = cv.resize(kernel, None, fx = rate, fy = rate, interpolation = cv.INTER\_NEAREST)

plt.imshow(kernel)

plt.title('kernel')

plt.show()

input\_image = cv.resize(input\_image, None, fx = rate, fy = rate, interpolation = cv.INTER\_NEAREST)

plt.imshow(input\_image)

plt.title('input\_image')

plt.show()

output\_image = cv.resize(output\_image, None , fx = rate, fy = rate, interpolation = cv.INTER\_NEAREST)

plt.imshow(output\_image)

plt.title('output\_image')

plt.show()

**Output** :

|  |  |
| --- | --- |
| Fig.: Input Image | Fig.: Hit and Miss of Image |

**Boundary extraction**

import numpy as np

import cv2

from matplotlib import pyplot as plt

image = cv2.imread('letter\_A.jpg',0)

retVal,mask = cv2.threshold(image,155,255,cv2.THRESH\_BINARY\_INV)

kernel = np.ones((7,7),np.uint8)

gradient = cv2.morphologyEx(mask, cv2.MORPH\_GRADIENT, kernel)

titles = ['Original Image',"Binary Image",'Morphological gradient']

images = [image,mask,gradient]

plt.figure(figsize=(13,5))

for i in range(3):

plt.subplot(1,3,i+1)

plt.imshow(images[i],'gray')

plt.title(titles[i])

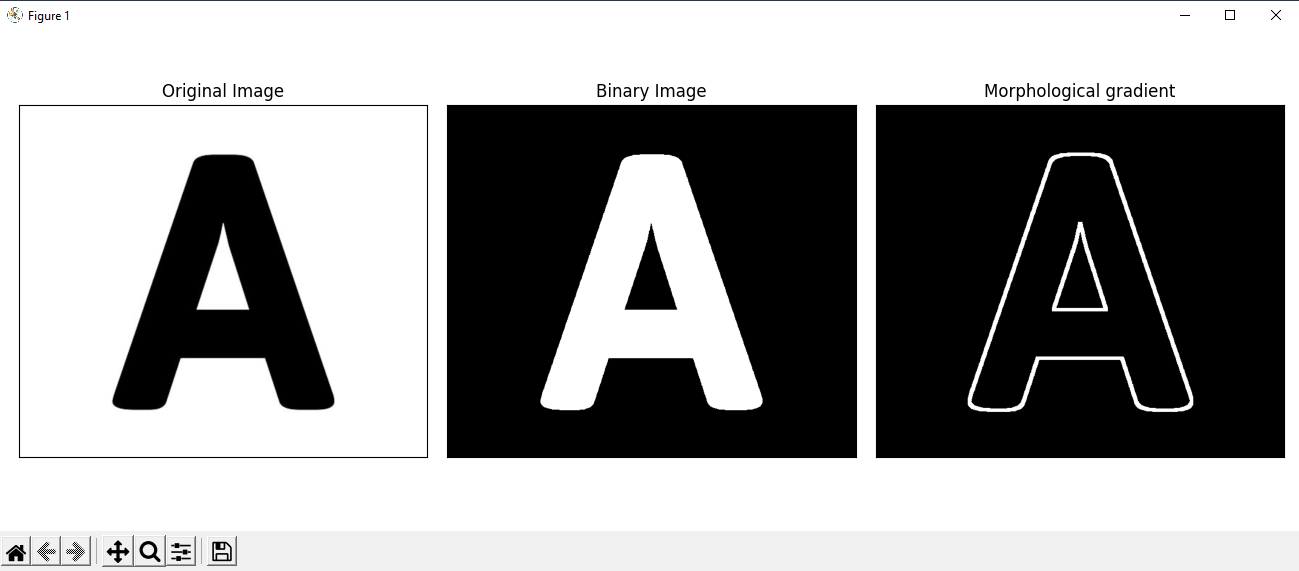
plt.xticks([])

plt.yticks([])

plt.tight\_layout()

plt.show()

**Output:**

****

**EXPERIMENT NO-8**

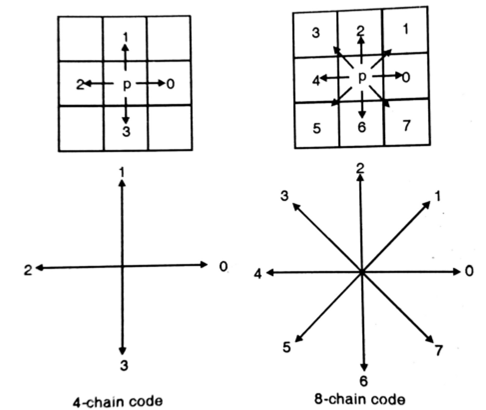
**Aim:** Write a program in python to find Chain code for 2-D Line.

**Software:** Python.

**Theory:** Chain code is a lossless compression technique used for representing an object in images. The co-ordinates of any continuous boundary of an object can be represented as a string of numbers where each number represents a particular direction in which the next point on the connected line is present. One point is taken as the reference/starting point and on plotting the points generated from the chain, the original figure can be re-drawn.

Chain codes are used to represent the binary by a connected sequence of straight –line segments. This represented is based on 4-connectivity and 8-connectivity of the segments.

The chain code works best with binary images and is a concise way of representing a shape contour. The chain code direction convention is given below:



As an edge is traced from its beginning point to the end point the direction that must be taken to move from one pixel to the next is given by the number represented in either the 4-chain code or the 8-chain code. As an edge can be completely described in terms of its starting coordinate and its sequence of chain codes descriptors. Of the two chain codes, the 4-chain is easier requiring only four different code values.

**Task :**

1. Read the image.
2. Perform Chain Code for 2D Line
3. Display the result

**Result Analysis and Conclusion**:

**Post Experiment Questions:**

* How does chain code represent the boundary of an object in a binary image?
* Discuss one advantage of using chain code over other methods for representing object boundaries in images.

**Code:**

# chain code

codeList = [5, 6, 7, 4, -1, 0, 3, 2, 1]

# This function generates the chaincode

# for transition between two neighbour points

def getChainCode(x1, y1, x2, y2):

dx = x2 - x1

dy = y2 - y1

hashKey = 3 \* dy + dx + 4

return codeList[hashKey]

'''This function generates the list of

chaincodes for given list of points'''

def generateChainCode(ListOfPoints):

chainCode = []

for i in range(len(ListOfPoints) - 1):

a = ListOfPoints[i]

b = ListOfPoints[i + 1]

chainCode.append(getChainCode(a[0], a[1], b[0], b[1]))

return chainCode

'''This function generates the list of points for

a straight line using Bresenham's Algorithm'''

def Bresenham2D(x1, y1, x2, y2):

ListOfPoints = []

ListOfPoints.append([x1, y1])

xdif = x2 - x1

ydif = y2 - y1

dx = abs(xdif)

dy = abs(ydif)

if(xdif > 0):

xs = 1

else:

xs = -1

if (ydif > 0):

ys = 1

else:

ys = -1

if (dx > dy):

# Driving axis is the X-axis

p = 2 \* dy - dx

while (x1 != x2):

x1 += xs

if (p >= 0):

y1 += ys

p -= 2 \* dx

p += 2 \* dy

ListOfPoints.append([x1, y1])

else:

# Driving axis is the Y-axis

p = 2 \* dx-dy

while(y1 != y2):

y1 += ys

if (p >= 0):

x1 += xs

p -= 2 \* dy

p += 2 \* dx

ListOfPoints.append([x1, y1])

return ListOfPoints

def DriverFunction():

(x1, y1) = (-9, -3)

(x2, y2) = (10, 1)

ListOfPoints = Bresenham2D(x1, y1, x2, y2)

chainCode = generateChainCode(ListOfPoints)

chainCodeString = "".join(str(e) for e in chainCode)

print ('Chain code for the straight line from', (x1, y1),

'to', (x2, y2), 'is', chainCodeString)

DriverFunction()

**Output:**

**EXPERIMENT NO-9**

**Aim:** Write a program in python to apply Support Vector Machine to split data set.

**Software:** Python

**Theory:**

Linear regression is a statistical technique used to model the relationship between a dependent variable and one or more independent variables. It assumes that the relationship between the variables is linear, which means that the change in the dependent variable is proportional to the change in the independent variable(s). The goal of linear regression is to find the best-fit line through the data points, which can then be used to predict the value of the dependent variable for new values of the independent variable(s). This is done by minimising the sum of the squared differences between the predicted values and the actual values of the dependent variable. Linear regression is widely used in fields such as economics, finance, and social sciences to understand the relationship between variables and make predictions about future outcomes.

**Discriminative Model:** SVM is a discriminative model designed for classification and regression analysis. It aims to find the optimal hyperplane that best separates data points into different classes.

**Hyperplane:**

* **Definition:** In SVM, a hyperplane is a decision boundary that maximally separates the data points of different classes in feature space.
* **Optimality:** SVM seeks the hyperplane that maximizes the margin, which is the distance between the hyperplane and the nearest data points (support vectors)**Margin:**
* **Margin Maximization:** The margin is crucial in SVM as it represents the distance between the decision boundary and the closest data points from each class. SVM aims to maximize this margin to improve the model's generalization to unseen data.

**Support Vectors:**

* **Definition:** Support vectors are the data points that lie closest to the decision boundary (hyperplane) and play a key role in determining the optimal hyperplane.
* **Influence on Hyperplane:** The position of support vectors influences the orientation and position of the decision boundary.

**Kernel Trick:**

* **Non-Linear Transformations:** SVM can handle non-linear relationships betweenfeatures and classes by employing the kernel trick. It implicitly maps the input data into a higher-dimensional space where a linear hyperplane can effectively separate classes.

**Regularization Parameter (C):**

* **Trade-off:** SVM introduces a regularization parameter (C) that balances the desire for a large margin and the cost of misclassifying training points.
* **Adjusting C:** A smaller C encourages a wider margin but allows more misclassifications, while a larger C results in a narrower margin but fewer misclassifications.

**Task:**

1. Read the image.
2. Perform linear regression
3. Display the result

**Result Analysis and Conclusion**:

**Post Experiment Questions:**

* What is the primary objective of linear regression using SVM.
* Explain how does SVM differ from classification tasks?

**Code:**

import numpy as np

import matplotlib.pyplot as plt

from google.colab import files

import pandas as pd

# Importing the dataset

uploaded = files.upload()

dataset = pd.read\_csv('/content/salary\_data.csv')

X = dataset.iloc[:, :-1].values #get a copy of dataset exclude last column

y = dataset.iloc[:, 1].values #get array of dataset in column 1st

dataset

# Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=1/3, random\_state=0)

# Fitting Simple Linear Regression to the Training set

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X\_train, y\_train)

# Visualizing the Training set results

viz\_train = plt

viz\_train.scatter(X\_train, y\_train, color='red')

viz\_train.plot(X\_train, regressor.predict(X\_train), color='blue')

viz\_train.title('Salary VS Experience (Training set)')

viz\_train.xlabel('Year of Experience')

viz\_train.ylabel('Salary')

viz\_train.show()

# Visualizing the Test set results

viz\_test = plt

viz\_test.scatter(X\_test, y\_test, color='red')

viz\_test.plot(X\_train, regressor.predict(X\_train), color='blue')

viz\_test.title('Salary VS Experience (Test set)')

viz\_test.xlabel('Year of Experience')

viz\_test.ylabel('Salary')

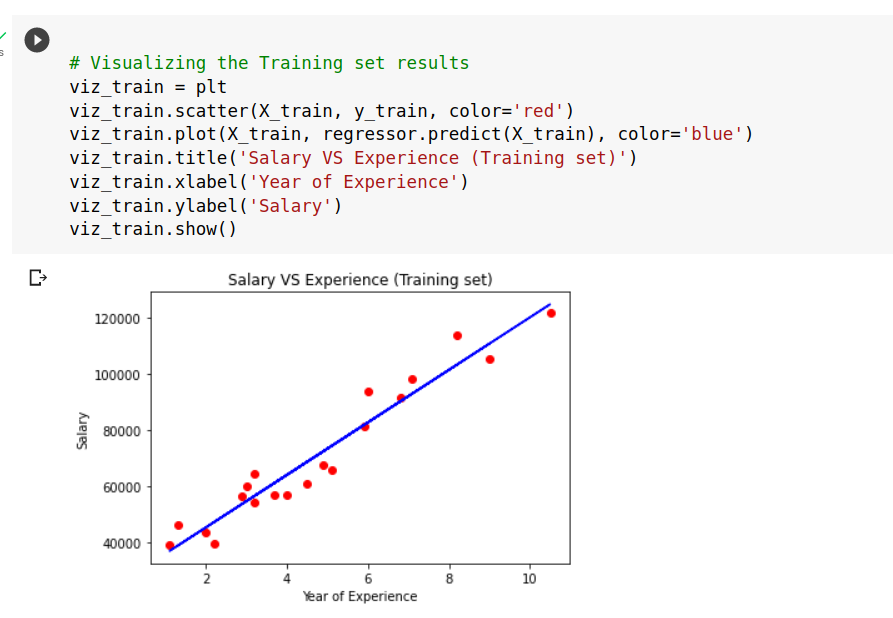
viz\_test.show()

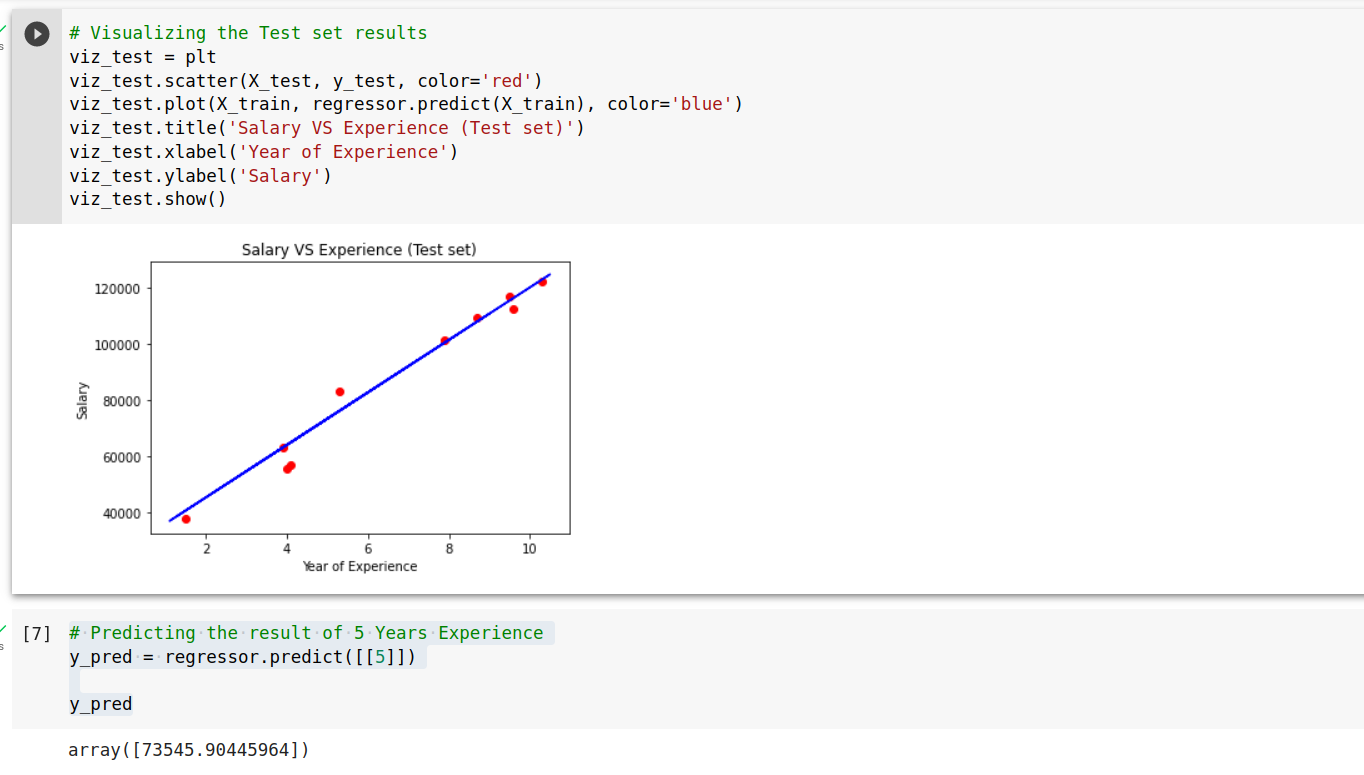
# Predicting the result of 5 Years Experience

y\_pred = regressor.predict([[5]])

y\_pred

**Output:**





**EXPERIMENT NO. 10**

**Aim:** Write a program in python to perform pattern clustering using K-means algorithm.

**Software**: Python

**Theory:** K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.The k-means clustering algorithm mainly performs two tasks:

* Determines the best value for K center points or centroids by an iterative process.
* Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters. The below diagram explains the working of the K-means Clustering Algorithm:



**Result Analysis and Conclusion**:

**Post Experiment Questions:**

* What is the main objective of the K-means clustering algorithm during clustering process?
* How does the choice of K influence the quality and interpretability of the resulting clusters?

**Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

#%matplotlib inline

X= -2 \* np.random.rand(100,2)

X1 = 1 + 2 \* np.random.rand(50,2)

X[50:100, :] = X1

plt.scatter(X[ : , 0], X[ :, 1], s = 50, c = 'b')

plt.show()

from sklearn.cluster import KMeans

Kmean = KMeans(n\_clusters=2)

Kmean.fit(X)

print(Kmean.cluster\_centers\_)

plt.scatter(X[ : , 0], X[ : , 1], s =50, c='b')

plt.scatter(-0.94665068, -0.97138368, s=200, c='g', marker='s')

plt.scatter(2.01559419, 2.02597093, s=200, c='r', marker='s')

plt.show()

print(Kmean.labels\_)

sample\_test=np.array([-3.0,-3.0])

second\_test=sample\_test.reshape(1, -1)

print(second\_test)

print(Kmean.predict(second\_test))

**Output:**

|  |  |
| --- | --- |
|  |  |

Fig. K- means Classification

 centroids:

[[ 1.94125027 2.02922224]

[-1.0424002 -0.84648622]]

K-means algorithm code:

[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

**EXPERIMENT NO. 11**

**Aim:** Write a program in python to apply MP Model to train network.

**Software:** Python

**Theory:** The McCulloch-Pitts model is a simple computational model of a neuron that was proposed in the 1940s by Warren McCulloch and Walter Pitts. It was the first attempt to mathematically model the behavior of a biological neuron. The model consists of binary inputs, weights, and a threshold, and produces a binary output based on the sum of the weighted inputs. It is based on the assumption that the behavior of a neuron can be reduced to a simple binary decision rule. Although the model is simplistic and cannot capture the full complexity of real neurons, it provided a foundation for the development of more sophisticated neural network models that can solve complex problems.

**Components of the McCulloch-Pitts Model:**

1. **Neuron Representation:**
   * Neurons are represented as binary units, either firing (outputting 1) or not firing (outputting 0).
2. **Binary Threshold Logic:**
   * Neurons employ a binary threshold logic function. The neuron fires if the weighted sum of its inputs exceeds a certain threshold; otherwise, it remains inactive.
3. **Weights and Inputs:**
   * Each input to a neuron is associated with a weight, representing the strength of the connection. The weighted sum of inputs is compared to the threshold to determine the neuron's output.
4. **Connections and Architecture:**
   * Neurons are connected in a network, forming a directed graph. Connections between neurons have associated weights.

**Contributions to Neural Network Theory:**

1. **Foundation for Neural Networks:**
   * The McCulloch-Pitts model laid the groundwork for the development of more advanced neural network models by introducing the fundamental concepts of neuron activation and connection weights.
2. **Binary Neuron Concept:**
   * The binary representation of neurons and the threshold logic concept have influenced subsequent models, serving as the basis for more sophisticated activation functions and learning algorithms.

**Task :**

1. Read the image.
2. Perform Classification
3. Display the result

**Result Analysis and Conclusion**:

**Post Experiment Questions:**

* What is the McCulloch-Pitts model, and how does it contribute to the foundational understanding of neural networks?
* What limitations does binary activation states (0 or 1) impose on the model's ability to capture complex information processing?

**Code:**

import numpy as np

np.random.seed(seed=0)

I = np.random.choice([0,1], 3)# generate random vector I, sampling from {0,1}

W = np.random.choice([-1,1], 3) # generate random vector W, sampling from {-1,1}

print(f'Input vector:{I}, Weight vector:{W}')

dot = I @ W

print(f'Dot product: {dot}')

def linear\_threshold\_gate(dot: int, T: float) -> int:

'''Returns the binary threshold output'''

if dot >= T:

return 1

else:

return 0

T = 1

activation = linear\_threshold\_gate(dot, T)

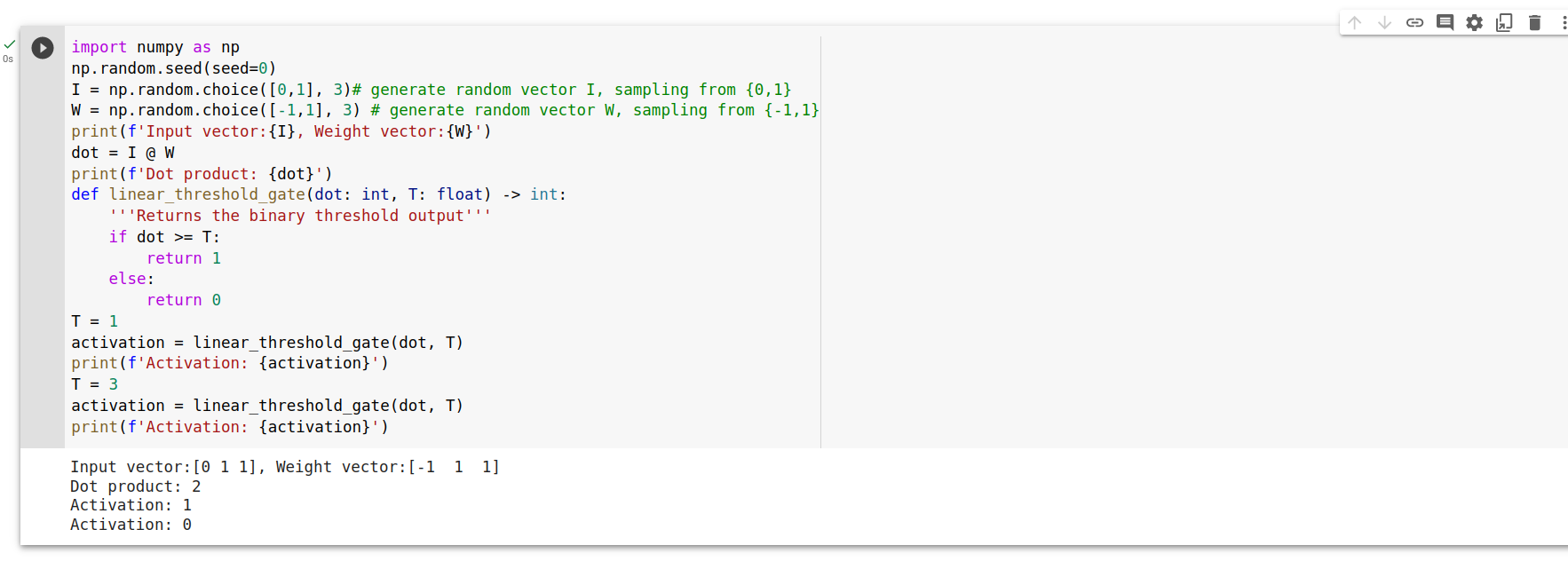
print(f'Activation: {activation}')

T = 3

activation = linear\_threshold\_gate(dot, T)

print(f'Activation: {activation}')

**Output:**

****

**EXPERIMENT NO 12**

**Aim:** Write a program in python to perform CNN using VGG16 model.

**Software:** Python

**Theory:** Convolutional Neural Networks (CNNs) are deep learning models that have been designed specifically for image recognition and analysis tasks. The VGG16 model is a popular pre-trained CNN architecture that was developed by the Visual Geometry Group (VGG) at the University of Oxford. The VGG16 model consists of 16 layers, including 13 convolutional layers and 3 fully connected layers, and has been trained on millions of images from the ImageNet dataset. This pre-trained model can be fine-tuned on a smaller dataset for specific image recognition tasks, providing high accuracy and performance. The VGG16 model has been widely used in a variety of computer vision applications, such as object detection, face recognition, and image classification.

**Activation Function:**

* + Rectified Linear Unit (ReLU) activation functions are used throughout the network after each convolutional and fully connected layer, except for the output layer.

**Normalization and Regularization:**

* + Batch normalization is applied after each convolutional and fully connected layer to improve training stability.
  + Dropout is used in the fully connected layers to prevent overfitting.

**Output Layer:**

* + The output layer has 1000 neurons with a softmax activation function, making VGG16 suitable for classifying images into 1000 categories as per the ImageNet dataset.

**Task :**

1. Read the image.
2. Perform Classification
3. Display the result

**Result Analysis and Conclusion**:

**Post Experiment Questions:**

* Highlight the key components, such as convolutional layers, pooling layers, and fully connected layers in VGG16 Model.
* How does the VGG16 architecture contribute to its effectiveness in image classification tasks?

**Code:**

from keras.applications.

vgg16 import VGG16

model = VGG16(weights='imagenet')

print(model.summary())

from tensorflow.keras.preprocessing import image

from tensorflow.keras.applications.vgg16 import preprocess\_input,decode\_predictions

import numpy as np

*in img\_path write path of the image for which the model will perform classification*

img\_path = '/content/dog1.jpg'

#There is an interpolation method to match the source size with the target size

#image loaded in PIL (Python Imaging Library)

img = image.load\_img(img\_path,color\_mode='rgb', target\_size=(224, 224))

display(img)

# Converts a PIL Image to 3D Numy Array

x = image.img\_to\_array(img)

x.shape

# Adding the fouth dimension, for number of images

x = np.expand\_dims(x, axis=0)

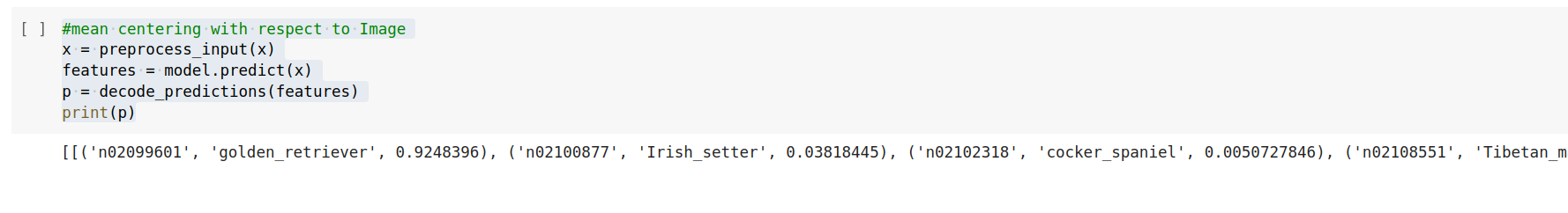
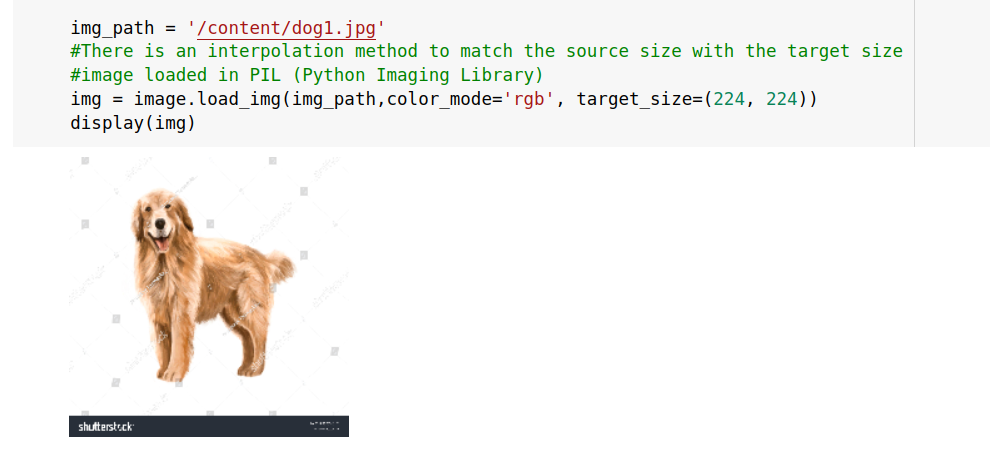
#mean centering with respect to Image

x = preprocess\_input(x)

features = model.predict(x)

p = decode\_predictions(features)

print(p)

**Output:**