# Write a program to display image matrix

from PIL import Image

from numpy import array

import cv2 as cv

im\_1 = cv.imread("/content/letter-7-transformed (1).png") ar = array(im\_1) ar

# write program to display Image Histogram

|  |
| --- |
| from PIL import Image  import matplotlib.pyplot as plt    # Open an image  image = Image.open("/content/Rotated-letter-7 (1).png")  # Convert the image to grayscale (optional)  image = image.convert("L")    # Calculate the histogram  histogram = image.histogram()    # Plot the histogram  plt.hist(histogram, bins=256, range=(0, 256), density=True, color='gray', alpha=0.7)  plt.title("Image Histogram")  plt.xlabel("Pixel Value")  plt.ylabel("Frequency")  plt.show() |

write program to find Histogram equalization and display that image

|  |
| --- |
| from PIL import Image import numpy as np  def histogram\_equalization(image):  # Convert the image to grayscale    # Convert the image to a NumPy array |
| img\_array = np.array(image)    # Calculate the histogram  hist, bins = np.histogram(img\_array, bins=256, range=(0, 256))  # Calculate the cumulative distribution function (CDF) cdf = hist.cumsum()    # Apply histogram equalization to the image  cdf\_min = cdf.min()  img\_eq = (cdf[img\_array] - cdf\_min) \* 255 / (cdf[-1] - cdf\_min)  # Convert the equalized NumPy array back to an image equalized\_image = Image.fromarray(np.uint8(img\_eq))  return equalized\_image    # Open an image  image = Image.open("/content/Screenshot 2023-10-16 192047.png")  # Perform histogram equalization  equalized\_image = histogram\_equalization(image)    # Display the original and equalized imag  equalized\_image |

# write program to smooth image using gaussian filter and display image

from PIL import Image, ImageFilter

# Open an image

image = Image.open("/content/Screenshot 2023-10-16 192047.png")

# Apply Gaussian smoothing to the image

smoothed\_image = image.filter(ImageFilter.GaussianBlur(radius=2))

# Display the original and smoothed images

image.show(title="Original Image")

smoothed\_image.show(title="Smoothed Image")

plt.show(smoothed\_image)

write program to find 1st order dertivative and display image and firsr order derivatice

|  |
| --- |
| import numpy as np  from scipy.ndimage import convolve import matplotlib.pyplot as plt import cv2    # Load an image  image = cv2.imread("your\_image.jpg", cv2.IMREAD\_GRAYSCALE)  # Compute the first-order derivative along the x and y axes dx = np.array([[-1, 0, 1]])  dy = dx.T  dx\_derivative = convolve(image, dx)  dy\_derivative = convolve(image, dy)    # Display the original image and its first-order derivatives plt.figure(figsize=(12, 6))  plt.subplot(131) plt.imshow(image, cmap="gray") plt.title("Original Image")  plt.subplot(132)  plt.imshow(dx\_derivative, cmap="gray") plt.title("First-Order Derivative (X-axis)")  plt.subplot(133)  plt.imshow(dy\_derivative, cmap="gray") plt.title("First-Order Derivative (Y-axis)")  plt.tight\_layout() plt.show() |

write program to find second order derivative and display image and second order derivative

|  |
| --- |
| import numpy as np  from scipy.ndimage import convolve import matplotlib.pyplot as plt import cv2    # Load an image  image = cv2.imread("your\_image.jpg", cv2.IMREAD\_GRAYSCALE)  # Compute the second-order derivative along the x and y axes d2x = np.array([[1, -2, 1]])  d2y = d2x.T  d2x\_derivative = convolve(image, d2x) |
| d2y\_derivative = convolve(image, d2y)    # Display the original image and its second-order derivatives plt.figure(figsize=(12, 6))  plt.subplot(131) plt.imshow(image, cmap="gray") plt.title("Original Image")  plt.subplot(132)  plt.imshow(d2x\_derivative, cmap="gray") plt.title("Second-Order Derivative (X-axis)")  plt.subplot(133)  plt.imshow(d2y\_derivative, cmap="gray") plt.title("Second-Order Derivative (Y-axis)")  plt.tight\_layout() plt.show() |

write program to determine Image gradient using Sobel operators

|  |
| --- |
| from PIL import Image, ImageFilter import numpy as np  import matplotlib.pyplot as plt    # Open an image  image = Image.open("your\_image.jpg")    # Convert the image to grayscale  image = image.convert("L")    # Apply the Sobel operators for gradient calculation  sobel\_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]) sobel\_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])    gradient\_x = image.filter(ImageFilter.Kernel((3, 3), sobel\_x)) gradient\_y = image.filter(ImageFilter.Kernel((3, 3), sobel\_y))  # Calculate the magnitude of the gradient  gradient\_magnitude = np.sqrt(np.array(gradient\_x)\*\*2 + np.array(gradient\_y)\*\*2)    # Display the original image and the gradient magnitude |
| plt.figure(figsize=(12, 6))  plt.subplot(131) plt.imshow(image, cmap="gray") plt.title("Original Image")  plt.subplot(132) plt.imshow(gradient\_x, cmap="gray") plt.title("Sobel X Gradient")  plt.subplot(133) plt.imshow(gradient\_y, cmap="gray") plt.title("Sobel Y Gradient")  plt.tight\_layout() plt.show()  plt.figure() plt.imshow(gradient\_magnitude, cmap="gray") plt.title("Gradient Magnitude")  plt.show() |