1. Write a Program to read a digital image. Split and display image into 4 quadrants, up, down, right, and left.

To write a program to read a digital image, split it into four quadrants and display them, we will need to use the Python Imaging Library (PIL). Here is a sample code that demonstrates how to accomplish this:

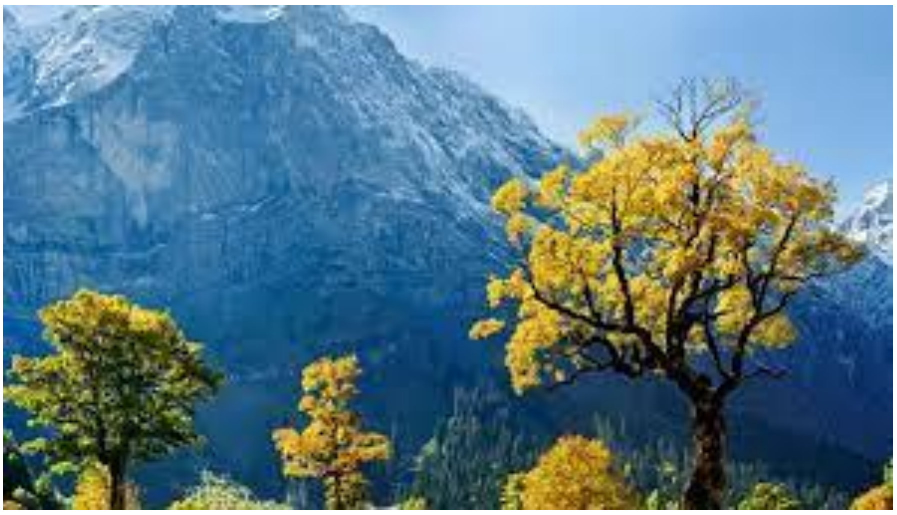
```
from PIL import Image
# Load the image
image_path = r'C:\Users\AIML\Documents\nature.jpg',
image = Image.open(image_path)

# Get the dimensions of the image
width, height = image.size

# Split the image into four quadrants
top_left = image.crop((0, 0, width//2, height//2))
top_right = image.crop((width//2, 0, width, height//2))
bottom_left = image.crop((0, height//2, width//2, height))
bottom_right = image.crop((width//2, height//2, width, height))

# Display the four quadrants
top_left.show()
top_right.show()
bottom_left.show()
bottom_right.show()
```

In the above code, we first import the Image module from PIL and then load the image from a specified path. Next, we get the width and height of the image using the size attribute.

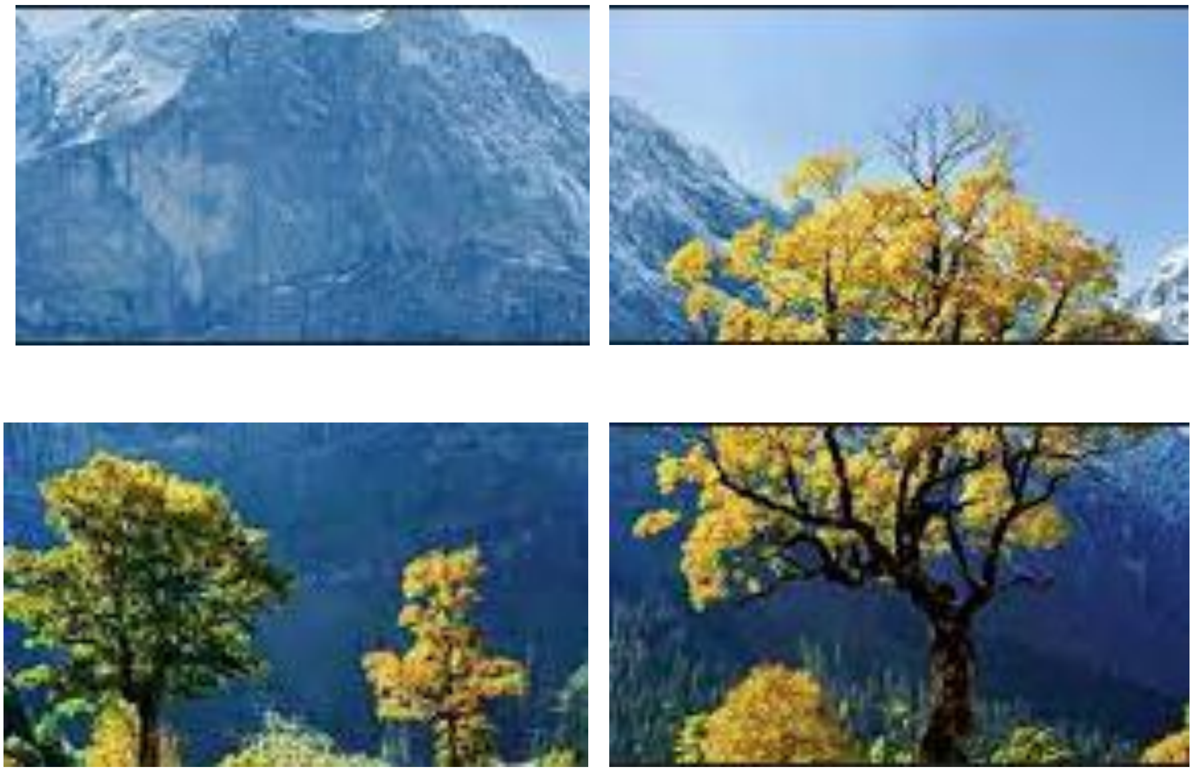
We then split the image into four quadrants using the crop method of the Image object. The crop method takes a tuple of four values representing the left, top, right, and bottom coordinates of the region to be cropped. We use integer division (//) to get the center point of the image.

Finally, we display the four quadrants using the show method of the Image object. This will open the four quadrants in separate windows.

Input Image:



Output:



2. Write a program to show rotation, scaling, and translation of an image.

To demonstrate rotation, scaling, and translation of an image using Python, we will use the Python Imaging Library (PIL). Here's a sample code that shows how to accomplish this:

```
from PIL import Image
# Load the image
image_path = r'C:\Users\AIML\Documents\lion.jpg'
image = Image.open(image_path)

# Rotate the image by 45 degrees
rotated_image = image.rotate(45)

# Scale the image by 2
scaled_image = image.resize((image.width * 2, image.height * 2))

# Translate the image by (100, 50) pixels
translated_image = image.transform(image.size, Image.AFFINE, (1, 0, 100, 0, 1, 50))

# Display the original, rotated, scaled, and translated images
image.show()
rotated_image.show()
scaled_image.show()
translated_image.show()
```

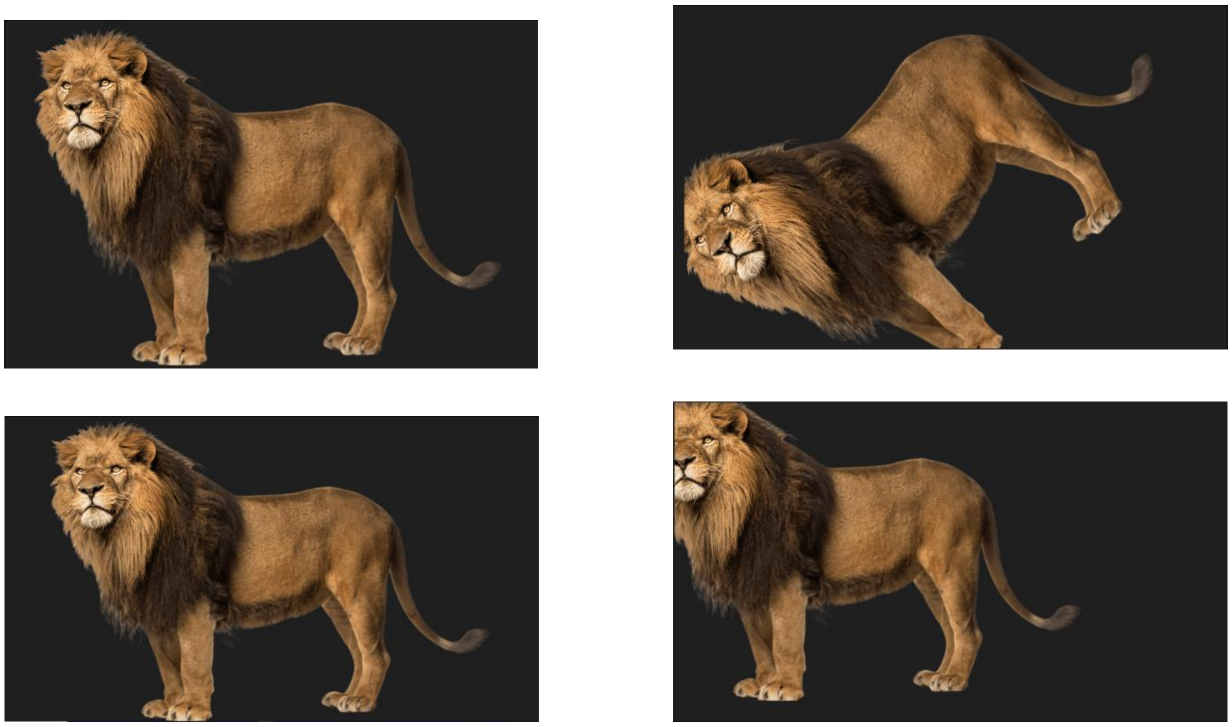
In the above code, we first import the Image module from PIL and then load the image from a specified path.
Next, we rotate the image by 45 degrees using the rotate method of the Image object. We store the result in the rotated_image variable.
After that, we scale the image by a factor of 2 using the resize method of the Image object. We multiply the original dimensions of the image by 2 and pass them as a tuple to the resize method. We store the result in the scaled_image variable.

Finally, we translate the image by (100, 50) pixels using the transform method of the Image object. We pass three arguments to the transform method: the size of the output image, the transformation matrix (in this case, an affine transformation matrix), and a tuple representing the translation vector. We store the result in the translated_image variable.

Finally, we display the original, rotated, scaled, and translated images using the show method of the Image object. This will open each image in a separate window.

Input Image:

Output:



3. Read an image, first apply erosion to the image and then subtract the result from the original. Demonstrate the difference in the edge image if you use dilation instead of erosion.

```
import cv2
import numpy as np

# Load the image in grayscale
image_path = r'C:\Users\AIML\Documents\butterfly.jpg',
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Define the kernel size for erosion and dilation
kernel_size = 5

# Apply erosion to the image
eroded_image = cv2.erode(image, np.ones((kernel_size,kernel_size),np.uint8))

# Apply dilation to the image
dilated_image = cv2.dilate(image, np.ones((kernel_size,kernel_size),np.uint8))

# Subtract the eroded image from the original
eroded_subtracted = cv2.absdiff(image, eroded_image)

# Subtract the dilated image from the original
dilated_subtracted = cv2.absdiff(image, dilated_image)

# Display the original, eroded-subtracted, and dilated-subtracted images
cv2.imshow('image',image)
cv2.imshow('eroded_subtracted', eroded_subtracted)
cv2.imshow('dilated_subtracted', dilated_subtracted)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
2. # Python program to demonstrate erosion and dilation of images.
import cv2
import numpy as np

# Reading the input image
img = cv2.imread(r'C:\Users\AIML\Documents\bu.jpg', 0)

# Taking a matrix of size 5 as the kernel
kernel = np.ones((5, 5), np.uint8)

# The first parameter is the original image,
# kernel is the matrix with which image is
# convolved and third parameter is the number
# of iterations, which will determine how much
# you want to erode/dilate a given image.
img_erosion = cv2.erode(img, kernel, iterations=1)
img_dilation = cv2.dilate(img, kernel, iterations=1)

cv2.imshow('Input', img)
cv2.imshow('Erosion', img_erosion)
cv2.imshow('Dilation', img_dilation)

cv2.waitKey(0)
```

In the above code, we first import the necessary modules: OpenCV and NumPy. We then load the image in grayscale using the imread function of OpenCV.

Next, we define the kernel size for erosion and dilation. The kernel size determines the extent of the erosion or dilation effect on the image.

We then apply erosion and dilation to the image using the erode and dilate functions of OpenCV, respectively. We pass a kernel of ones to both functions to achieve the maximum erosion and dilation effect.

After that, we subtract the eroded image from the original using the absdiff function of OpenCV. This results in an edge image with thicker edges compared to the original.

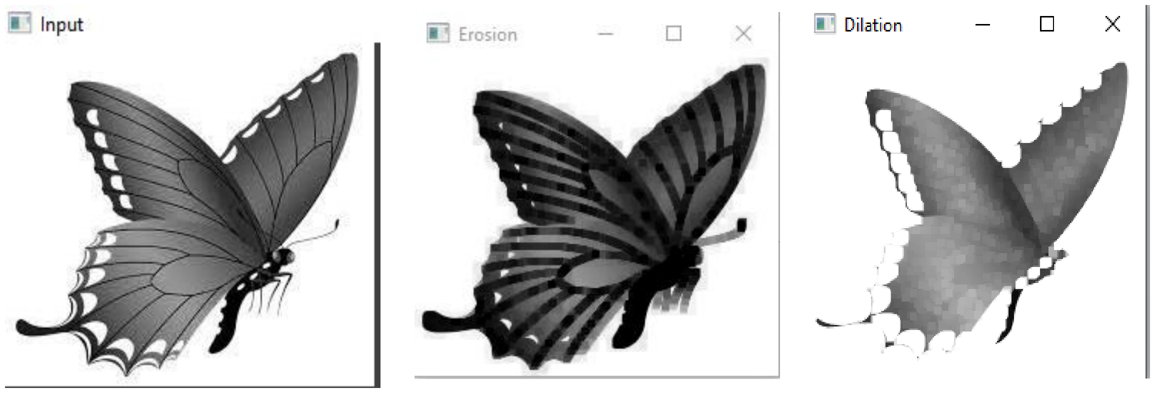
Finally, we subtract the dilated image from the original using the absdiff function of OpenCV. This results in an edge image with thinner edges compared to the original.

We display the original, eroded-subtracted, and dilated-subtracted images using the imshow function of OpenCV. This will open each image in a separate window.

In summary, erosion removes the edges of the image while dilation adds to the edges. Subtracting the eroded image from the original results in thicker edges, while subtracting the dilated image results in thinner edges.

Input Image:

Output:



4. Read an image and extract and display low-level features such as edges, textures using filtering Techniques.

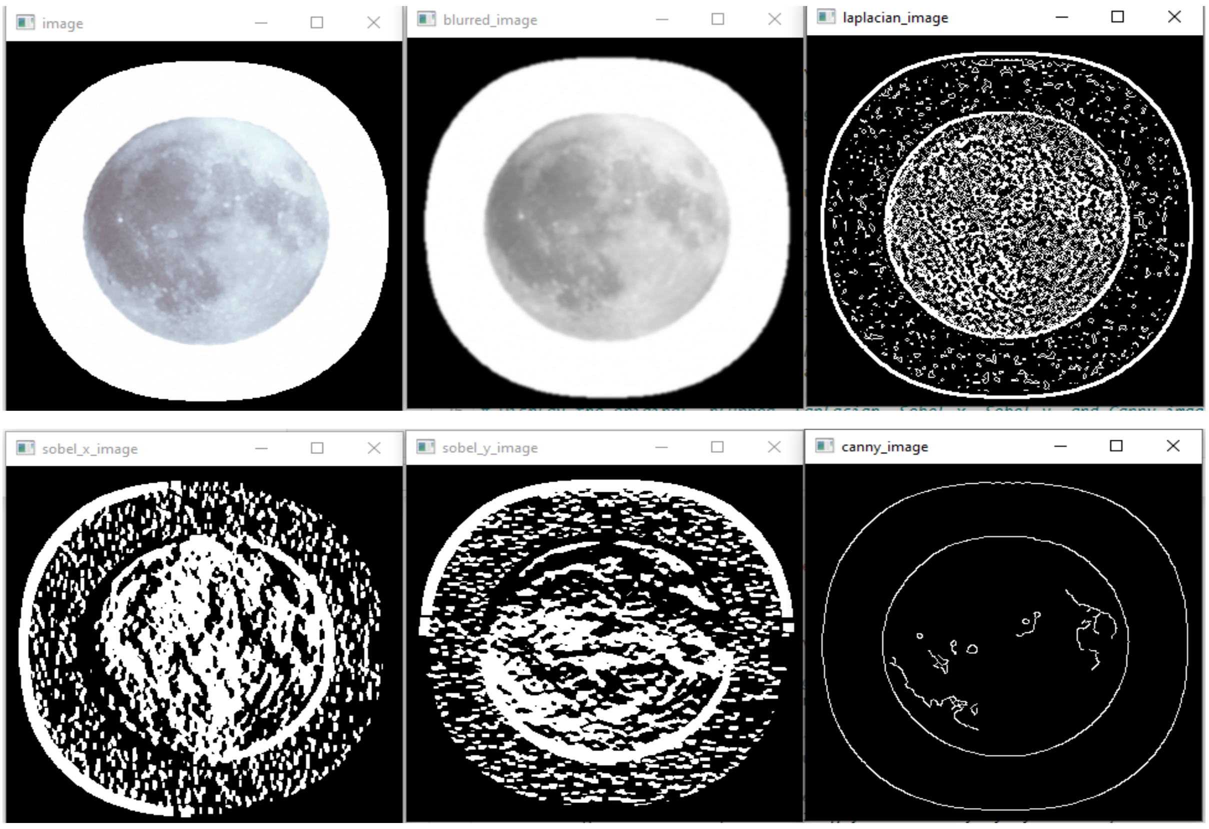
To extract low-level features such as edges and textures from an image using filtering techniques, we can use Python and the OpenCV library. Here's a sample code that shows how to apply various filters to an image to extract edges and textures:

```
import cv2
import numpy as np

# Load the image

image_path = r'C:\Users\AIML\Documents\moon.png'
image = cv2.imread(image_path)

# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply a Gaussian blur to the image
blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)

# Apply a Laplacian filter to the image
laplacian_image = cv2.Laplacian(blurred_image, cv2.CV_64F)

# Apply a Sobel filter to the image in the x direction
sobel_x_image = cv2.Sobel(blurred_image, cv2.CV_64F, 1, 0, ksize=5)

# Apply a Sobel filter to the image in the y direction
sobel_y_image = cv2.Sobel(blurred_image, cv2.CV_64F, 0, 1, ksize=5)

# Apply a Canny edge detector to the image
canny_image = cv2.Canny(blurred_image, 30, 100)
```

```
# Display the original, blurred, Laplacian, Sobel_x, Sobel_y, and Canny images
  cv2.imshow('image',image)
  cv2.imshow('blurred_image',blurred_image)
  cv2.imshow('laplacian_image', laplacian_image)
  cv2.imshow('sobel_x_image', sobel_x_image)
  cv2.imshow('sobel_y_image', sobel_y_image)
  cv2.imshow('canny_image', canny_image)
  cv2.waitKey(0)
  cv2.destroyAllWindows()
```

Input Image:

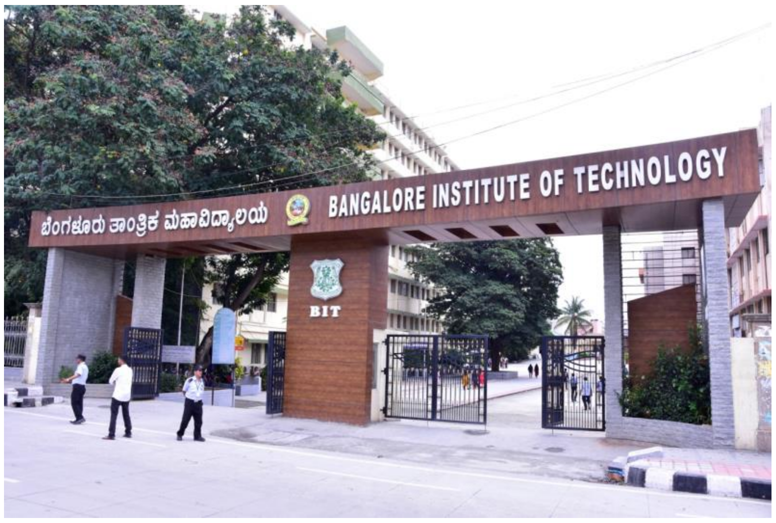
Output:



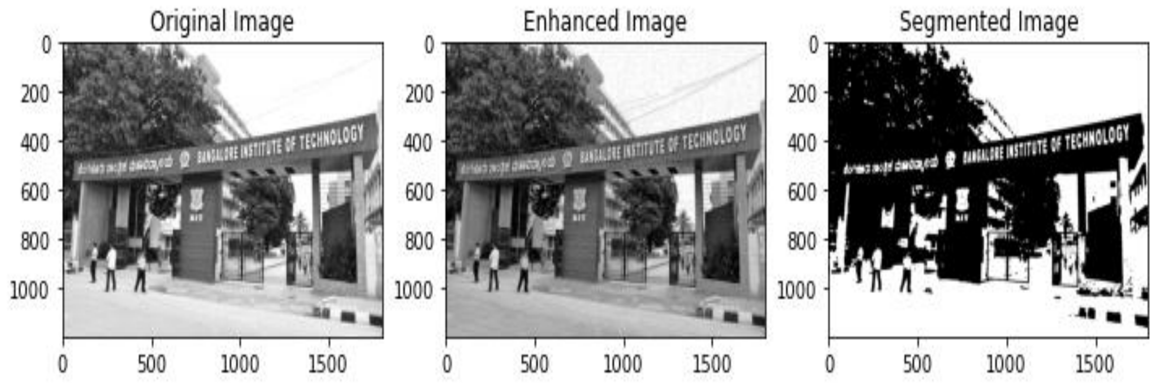
5. Demonstrate enhancing and segmenting low contrast 2D images.

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

# Load the image
img = cv2.imread(r'C:\Users\AIML\Documents\bit.jpg', cv2.IMREAD_GRAYSCALE)

# Apply histogram equalization
img_eq = cv2.equalizeHist(img)
# Apply Otsu's thresholding
_, thresh = cv2.threshold(img_eq, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OT
SU)

# Perform morphological operations to remove noise and fill in gaps
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
closed = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel)
opened = cv2.morphologyEx(closed, cv2.MORPH_OPEN, kernel)
# Display the images
fig, ax = plt.subplots(1, 3, figsize=(12, 4))
ax[0].imshow(img, cmap='gray')
ax[0].set_title('Original Image')
ax[1].imshow(img_eq, cmap='gray')
ax[1].set_title('Enhanced Image')
ax[2].imshow(opened, cmap='gray')
ax[2].set_title('Segmented Image')
plt.show()
```

Input Image:


ಬೆಂಗಳೂರು ತಾಂತ್ರಿಕ ಮಹಾವಿದ್ಯಾಲಯ
BANGALORE INSTITUTE OF TECHNOLOGY
BIT

Output:


Original Image
Enhanced Image
Segmented Image