UNIVERSITY OF CALCUTTA

B.Sc Semester 5 Honours Examination-2024

Under CBCS System

University Registration Number: 012-1114-1749-21

University Roll Number: 213012-21-0396

Subject: Computer Science

Subject Code: CMSA

Paper: Image Processing Lab Using Python

Paper Code: DSE-A1

Semester: 5

Year: 2024

INDEX

Q1) Write a program to read an image and rotate that image clockwise without using library function.

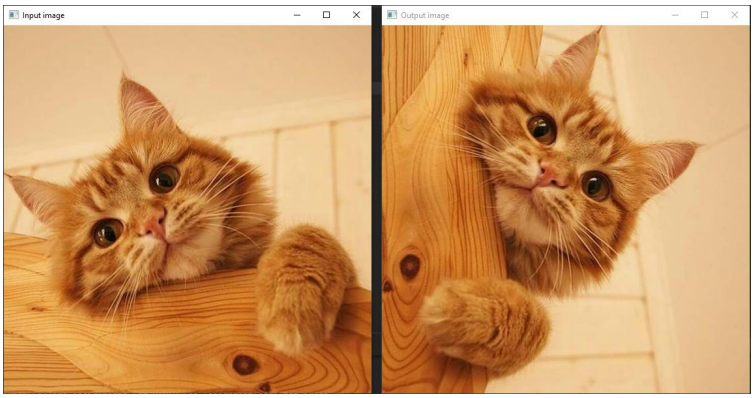
Problem Statement: The problem is to read an image and rotate it clockwise without using library function and display the resulting image.

Background Theory: To rotate an image clockwise, the algorithm involves reading the image and performing a 90-degree clockwise rotation by swapping the rows and columns. This rotation is achieved through nested loops that iterate over the pixels of the original image and assign them to the corresponding positions in the rotated image

Python Code:

```
import numpy as np
import cv2 as cv
# Read the input image
img = cv.imread("ass1/cat.jpg", 1)
cv.imshow("Input image", img)
h, w, c = img.shape
result = np.zeros([w, h, c], dtype=np.uint8)
# Rotate the image clockwise
for i in range(h):
    for j in range(w):
        result[j, h-1-i] = img[i, j]
# Display the rotated image
cv.imshow("Output image", result)
cv.waitKey(0)
cv.destroyAllWindows()
```

Input Image vs Output Image:



Conclusion/Observation: The provided code successfully accomplishes the task of rotating an image clockwise and displaying both the original and rotated images. Key observations from the output include:

- The color and texture features of both the input and output images remain the same.

- The output image is visibly rotated concerning the input image.

Q2) Write a program to find the histogram of a gray image and display the histogram.

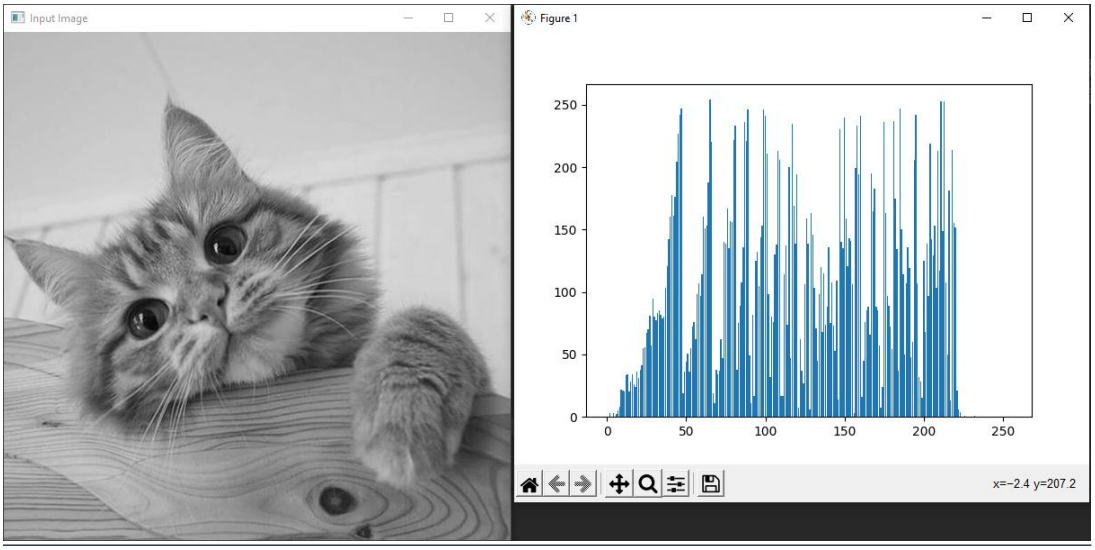
Problem Statement: The challenge is to write a program that finds and displays the histogram of a gray image.

Background Theory: The histogram of a gray image represents the distribution of pixel intensities. It is obtained by counting the frequency of each intensity level. The x-axis of the histogram represents pixel intensity values, and the y-axis represents the frequency of occurrence.

Python Code:

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
#function to form Histogram
def form_histogram(img):
    h,w=img.shape
    histogram=np.zeros([256],dtype=np.uint8)
    #calculate histogram
    for i in range(h):
        for j in range(w):
            frequency=img[i,j]
            histogram[frequency]+=1
    return histogram
#read the input image
image=cv.imread("ass2/cat.jpg",0)
#display the input image
cv.imshow("Input Image",image)
#calculate histogram
h=form_histogram(image)
#display the histogram
plt.bar(np.arange(len(h)),h)
plt.show()
cv.waitKey(0)
cv.destroyAllWindows()
```

Input Image vs Output Image:



Conclusion/Observation: The implemented code successfully computes and visualizes the histogram of a grayscale image. Observations from the generated output encompass:

- The histogram accurately portrays the distribution of pixel intensities in the grayscale image.
- The x-axis denotes pixel intensity values, while the y-axis represents the frequency of their occurrence

---

Q3) Write a program to enhance the image in the spatial domain using Image negative.

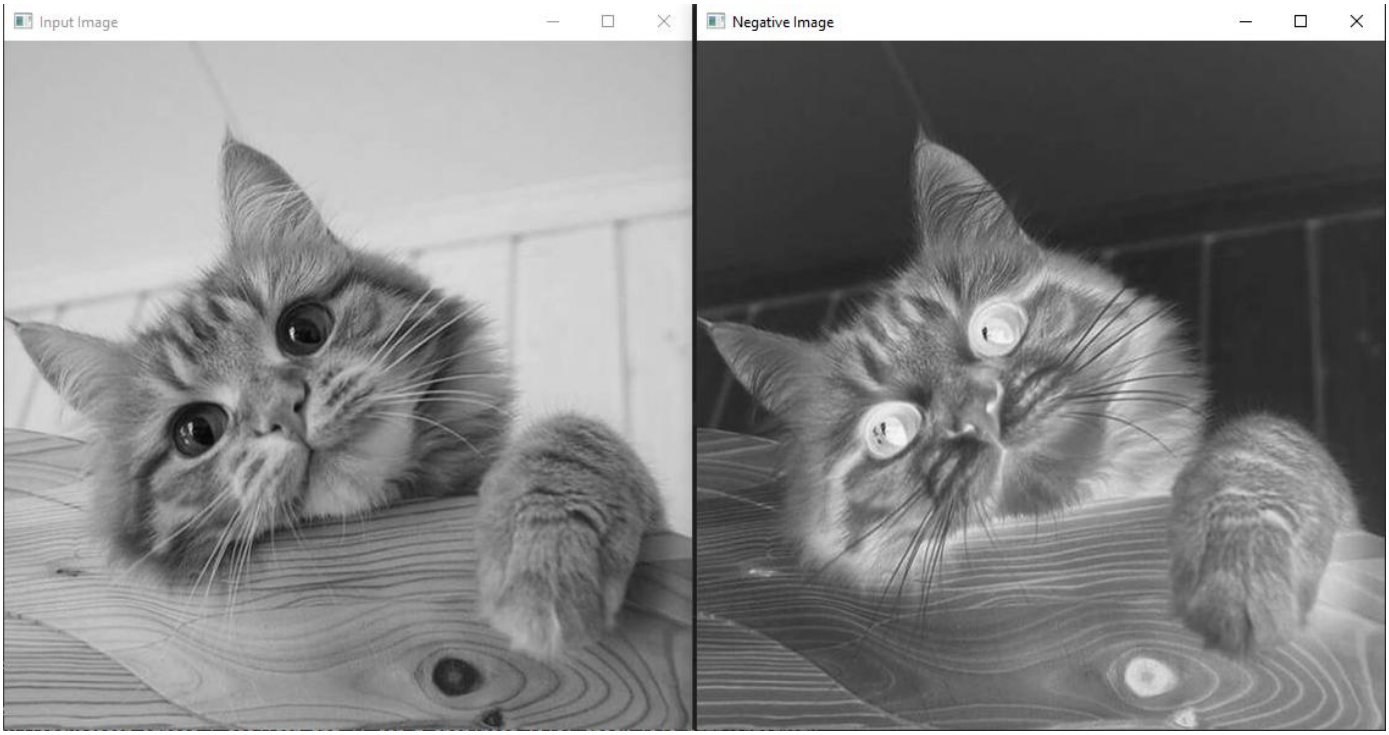
Problem Statement: The goal is to enhance an image in the spatial domain using the Image negative and visualize the transformation.

Background Theory: Image negative is a spatial domain operation where each pixel value in the image is subtracted from the maximum pixel value. This operation enhances the contrast of the image by inverting its intensity levels.

Python Code:

```
import cv2
import numpy as np
# Function to get negative image
def negative(image):
    h, w = image.shape[:2]
    # Creating a new matrix to store the negative image
    result = np.zeros_like(image, dtype=np.uint8)
    for i in range(h):
        for j in range(w):
            result[i, j] = 255 - image[i, j]
    return result
# Read the input image (assuming it is a grayscale image)
Img = cv2.imread("ass3\cat.jpg", 0)
# Display the input image
cv2.imshow("Input Image", img)
# Display the negative image
cv2.imshow("Negative Image", negative(img))
# Wait for a key press and close windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Input Image vs Output Image:



Conclusion/Observation:

The code for creating the image negative is effective in enhancing the spatial domain. Noteworthy observations from the output include:

- The image negative operation results in improved contrast by inverting the intensity levels.

- Both the original and negative images maintain the same color and texture features.

Q4) Write a program to enhance the image in the spatial domain using the histogram equalization method

Problem Statement: To enhance the given image using histogram equalization and display the input and output image.

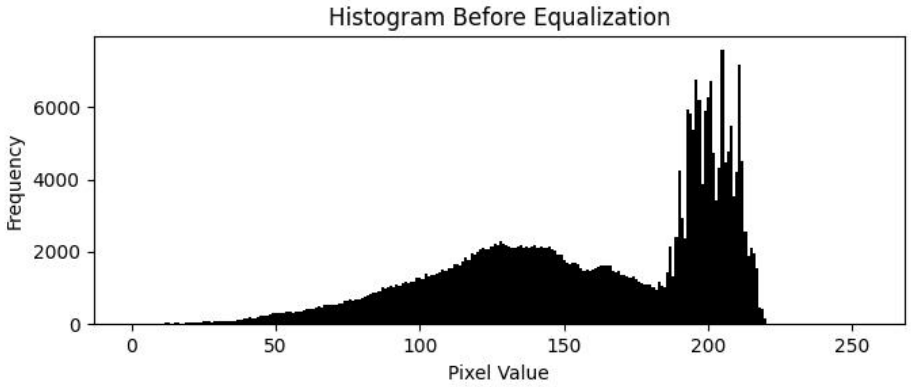
Background Theory: Histogram equalization is an image processing technique designed to enhance contrast by redistributing pixel intensities. It achieves this by calculating the cumulative distribution function (CDF) from the original histogram and then mapping pixel intensities to new values using this function. The goal is to create a more uniform distribution of intensities, resulting in improved visibility of details and features in both dark and bright regions of the image.

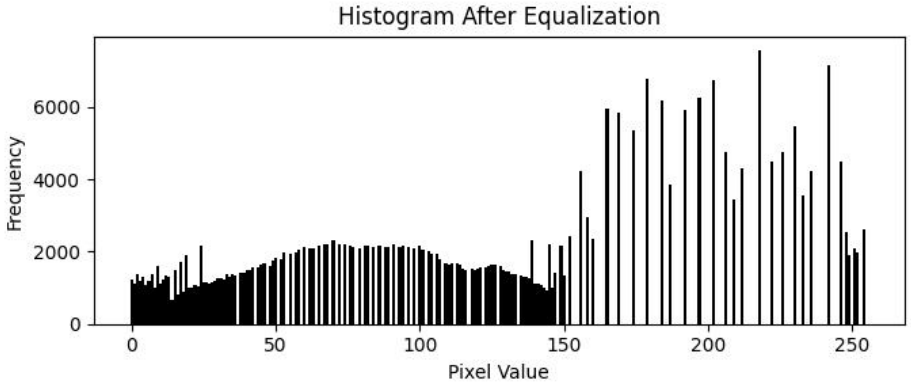
Python Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt  # Import Matplotlib library
def compute_histogram(image):
    histogram = np.zeros(256, dtype=int)
    h, w = image.shape[:2]
    for i in range(h):
        for j in range(w):
            intensity = image[i, j]
            histogram[intensity] += 1
    return histogram
def histogram_equalization(image):
    h, w = image.shape[:2]
    total_pixels = h * w
    hist = compute_histogram(image)
    # Compute cumulative distribution function (CDF) manually
    cdf = np.zeros(256, dtype=int)
    cdf[0] = hist[0]
    for i in range(1, 256):
        cdf[i] = cdf[i - 1] + hist[i]
    cdf_normalized = cdf * 255 / total_pixels
    equalized_image = np.zeros_like(image, dtype=np.uint8)
    for i in range(h):
        for j in range(w):
            equalized_image[i, j] = cdf_normalized[image[i, j]]
```

```
    return equalized_image
img = cv2.imread("ass4/cat.jpg", cv2.IMREAD_GRAYSCALE)
# Compute histograms before and after equalization
hist_before = compute_histogram(img)
equalized_img = histogram_equalization(img)
hist_after = compute_histogram(equalized_img)
# Plotting histograms using bar plots
plt.figure(figsize=(12, 6))
plt.subplot(2, 2, 1)
plt.title("Original Image")
plt.imshow(img, cmap="gray")
plt.axis("off")
plt.subplot(2, 2, 2)
plt.title("Histogram Before Equalization")
plt.bar(range(256), hist_before, color="black",width=1)
plt.xlabel("Pixel Value")
plt.ylabel("Frequency")
plt.subplot(2, 2, 3)
plt.title("Enhanced Image")
plt.imshow(equalized_img, cmap="gray")
plt.axis("off")
plt.subplot(2, 2, 4)
plt.title("Histogram After Equalization")
plt.bar(range(256), hist_after, color="black",width=1)
plt.xlabel("Pixel Value")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Input Image vs Output Image





Conclusion/Observation:

The code successfully enhances an image in the spatial domain through histogram equalization. Key observations from the output comprise:

- Histogram equalization redistributes pixel intensities, enhancing the overall contrast.

- The original and equalized images can be compared to appreciate the improved visibility of details in various intensity ranges.

Q5) Write a program to perform Image addition and display original and postprocessed images.

Problem Statement: Develop a program to perform Image addition, displaying both the original and postprocessed images.

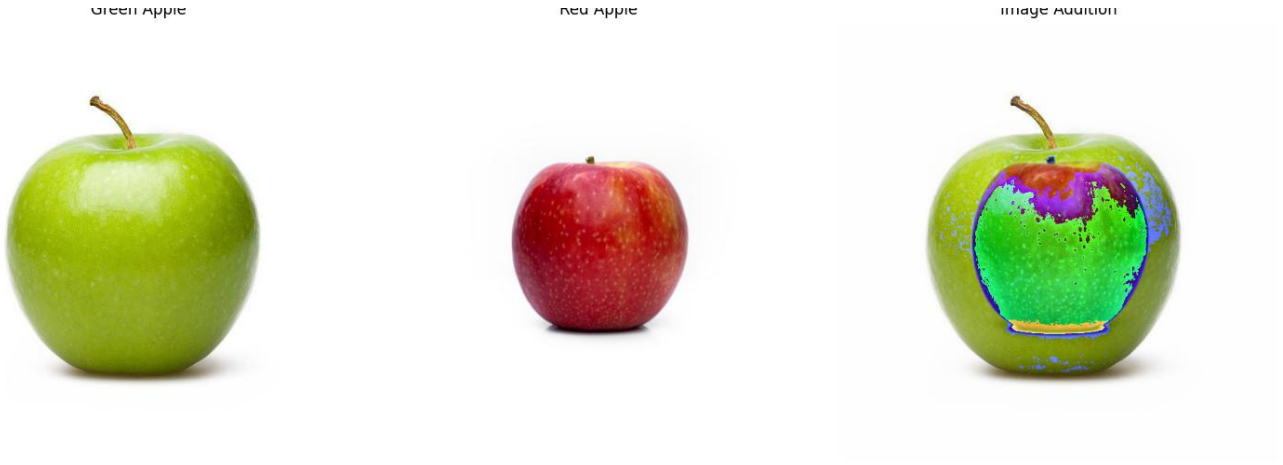
Background Theory: Image addition involves adding pixel values of corresponding positions in two images. This operation is often used for blending or combining two images. The resulting pixel values may need to be normalized to ensure they stay within the valid intensity range.

Python Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def make_square(image, pad_value=(255, 255, 255)):
    height, width = image.shape[:2]
    size = max(height, width)
    # Calculate padding values
    top = (size - height) // 2
    bottom = size - height - top
    left = (size - width) // 2
    right = size - width - left
    # Create a square image with padding
    padded_image = np.full((size, size, 3), pad_value, dtype=np.uint8)
    padded_image[top:top+height, left:left+width, :] = image
    return padded_image
# Load the images using cv2
green_image = cv2.imread("ass5/greenapple.jpg")
red_image = cv2.imread("ass5/redapple.jpg")
# Make both images square with white padding
green_image_square = make_square(green_image)
red_image_square = make_square(red_image)
# Ensure both images are in the BGR color space
green_image_square = cv2.cvtColor(green_image_square, cv2.COLOR_BGR2RGB)
red_image_square = cv2.cvtColor(red_image_square, cv2.COLOR_BGR2RGB)
# Resize images to a common size
common_size = (612, 612)
green_image_resized = cv2.resize(green_image_square, common_size)
red_image_resized = cv2.resize(red_image_square, common_size)
```

```
# Perform image addition and clip the result to [0, 255]
result_image = np.clip(green_image_resized + red_image_resized, 0, 255).astype(np.uint8)
# Display the original images and the result
plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plt.title("Green Apple")
plt.imshow(green_image_square)
plt.axis("off")
plt.subplot(1, 3, 2)
plt.title("Red Apple")
plt.imshow(red_image_square)
plt.axis("off")
plt.subplot(1, 3, 3)
plt.title("Image Addition")
plt.imshow(result_image)
plt.axis("off")
plt.tight_layout()
plt.show()
```

Input Image vs Output Image



Conclusion/Observation:

The code for image addition effectively combines two images, showcasing both the original and postprocessed results. Notable observations include:

- Pixel values of corresponding positions are added, producing a blended result.

- The original images and the result can be visually compared for a qualitative assessment.

Q6) Write a program to perform the Brightness suppression image enhancement methods.

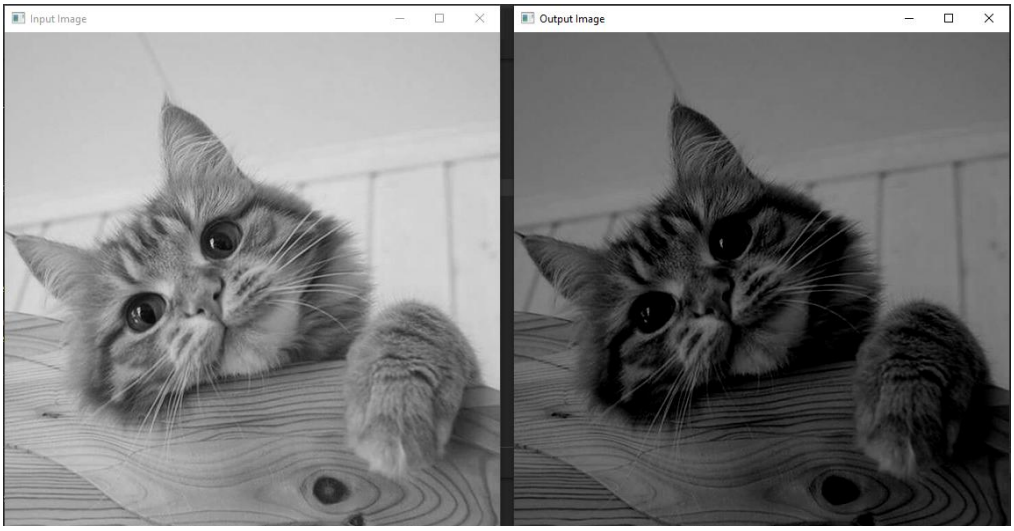
Problem Statement: Write a program to perform Brightness suppression image enhancement methods and illustrate the changes in the image

Background Theory: Brightness suppression methods aim to reduce the overall brightness of an image. This can be achieved through various techniques like gamma correction or adjusting the pixel values to make the image appear darker.

Python Code:

```
import numpy as np
import cv2 as cv
def intensity_suppression(img, factor):
    # Get the height and width of the image
    h, w = img.shape
    # Create an empty matrix for the result with the same size as the input image
    result = np.zeros_like(img, dtype=np.uint8)
    # Iterate through each pixel in the image
    for i in range(h):
        for j in range(w):
            # Apply suppression by subtracting the specified factor from the pixel intensity
            result[i, j] = max(0, img[i, j] - factor)
    return result
# Read the input grayscale image
image = cv.imread("ass6/cat.jpg", cv.IMREAD_GRAYSCALE)
# Display the original image
cv.imshow("Input Image", image)
# Apply intensity suppression with a specified factor (85 in this case)
suppressed_image = intensity_suppression(image, 85)
# Display the output image after intensity suppression
cv.imshow("Output Image", suppressed_image)
# Wait for a key press and close windows
cv.waitKey(0)
cv.destroyAllWindows()
```

Input Image vs Output Image:



Conclusion/Observation:

The code for brightness suppression successfully adjusts pixel values, resulting in a visually suppressed image. Key observations are:

- Brightness suppression methods, as applied, effectively reduce the overall brightness of the image.

- The color and texture features between the original and suppressed images remain similar

Q7) Write a program to add Gaussian noise and display both original and noisy images.

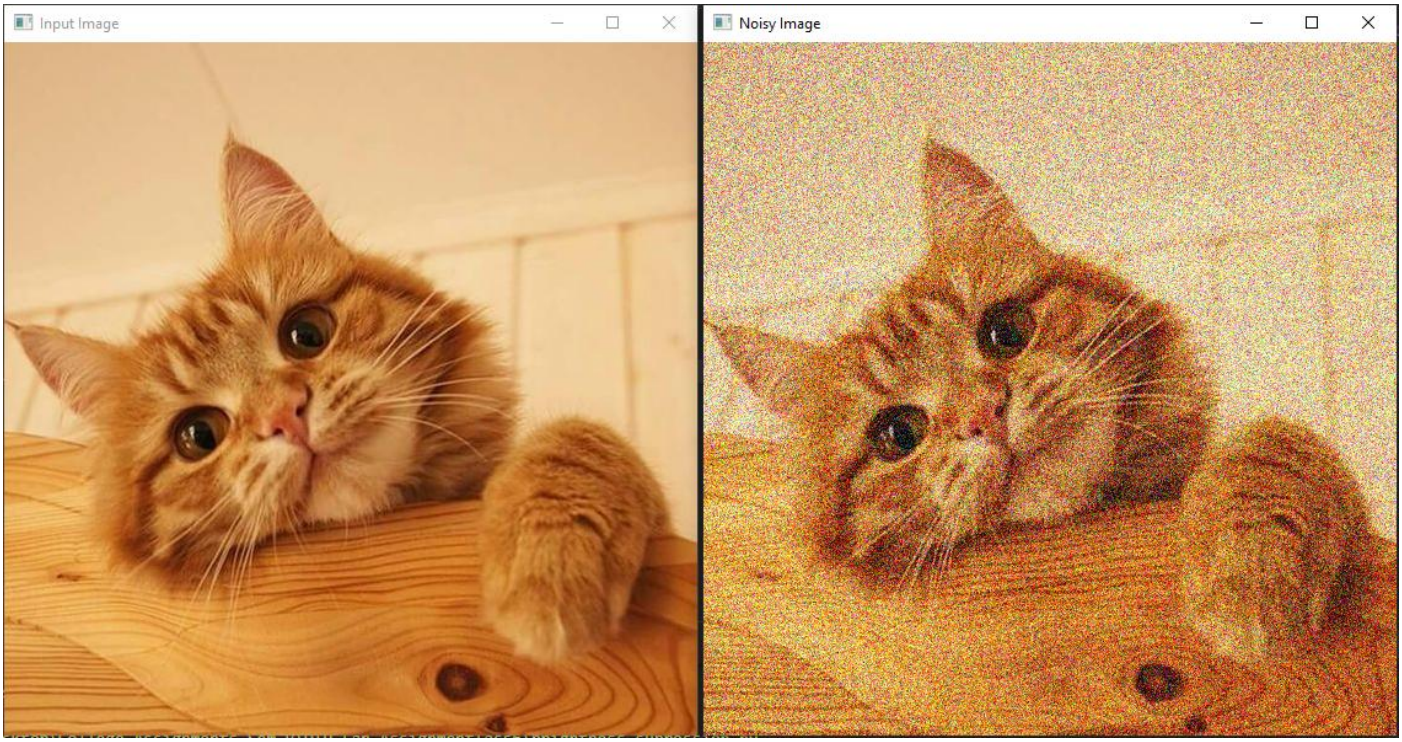
Problem Statement: Program to add Gaussian noise to an image and display both the input image and processed noisy image.

Background Theory: Gaussian noise is a type of statistical noise characterized by a Gaussian (normal) distribution of random values. In images, it appears as random variations with a bell-shaped intensity distribution. It can impact image quality, and noise reduction techniques are frequently applied to improve the visual appearance and analysis of images.

Python Code:

```
import numpy as np
import cv2 as cv
def add_gaussian_noise(img, mean=0, sigma=50):
    h, w, c = img.shape
    # Generate noise with the same shape as that of the image
    gauss_noise = np.random.normal(mean, sigma, (h, w, c))
    # Add noise to the image
    noisy_img = np.clip(img + gauss_noise,0,255).astype(np.uint8)
    return noisy_img
# Read the input image
image = cv.imread("ass7/cat.jpg")
# Display the input image
cv.imshow("Input Image", image)
# Invoke the function to add Gaussian noise
noisy_image = add_gaussian_noise(image)
# Display the noisy image
cv.imshow("Noisy Image", noisy_image)
# Wait for any key press and close windows
cv.waitKey(0)
cv.destroyAllWindows()
```

Input image vs Output image



Conclusion/Observation:

The code for adding Gaussian noise is proficient in introducing noise to the image. Observations include:

- Gaussian noise, when added, simulates real-world noise scenarios, affecting the overall image appearance.

- Both the original and noisy images maintain the same underlying color and texture features.

Q8) Write a program to enhance an image using mean filtering.

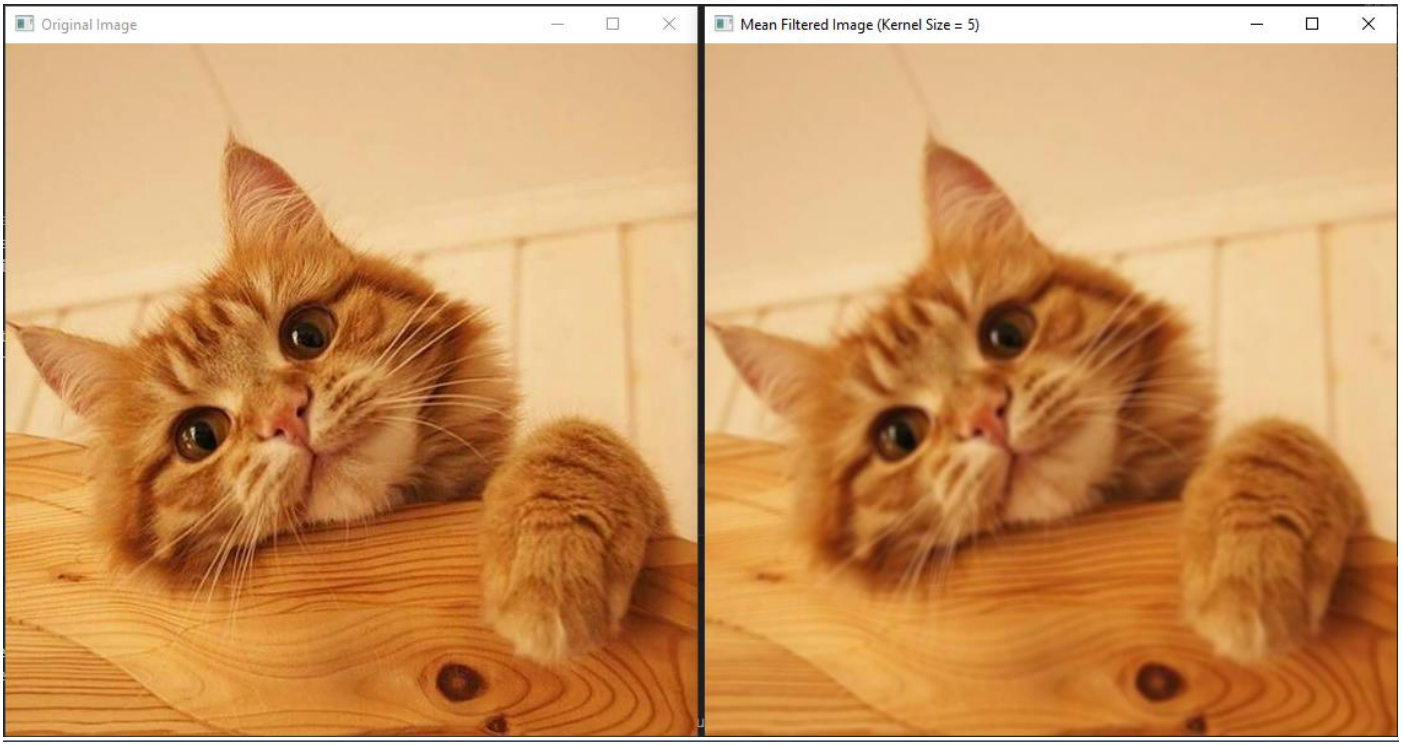
Problem Statement: Program to apply mean filter on a given image and display the input image and post processed image.

Background Theory: The mean filter in image processing is a spatial domain technique aimed at reducing noise and achieving image smoothing. It operates by replacing each pixel's intensity with the average value of its neighbouring pixels within a defined window. This process effectively reduces random noise in the image while introducing a blurring effect.

Python Code:

```
import cv2
import numpy as np
def apply_mean_filter(image, kernel_size):
    height, width, channels = image.shape
    result_image = np.zeros_like(image)
    for i in range(height):
        for j in range(width):
            for k in range(channels):
                # Calculate mean for each channel within the kernel
                start_i, start_j = max(0, i - kernel_size // 2), max(0, j - kernel_size // 2)
                end_i, end_j = min(height - 1, i + kernel_size // 2), min(width - 1, j + kernel_size // 2)
                # Calculate mean value
                mean_value = np.mean(image[start_i:end_i + 1, start_j:end_j + 1, k])
                result_image[i, j, k] = int(mean_value)
    return result_image
image = cv2.imread("ass8/cat.jpg")
cv2.imshow("Original Image", image)
# Specify the kernel size for the mean filter
kernel_size = 5
# Apply the mean filter
filtered_image = apply_mean_filter(image, kernel_size)
# Display the mean-filtered image
cv2.imshow(f"Mean Filtered Image (Kernel Size = {kernel_size})", filtered_image)
cv2.waitKey(0)
# Close windows
cv2.destroyAllWindows()
```

Input Image vs Output Image:



Conclusion/Observation:

This program successfully applies mean filter on an image and displays it. From the output image, we observe:

- Mean filter effectively reduces random noise in the image.

- The filter introduces a smoothing effect, making the overall appearance of the image more homogeneous.

Q9) Write a program to find the edge of a given image using the Prewitt operator.

Problem Statement: Problem to find the edge of the given image using Prewitt operator and display the input and output image.

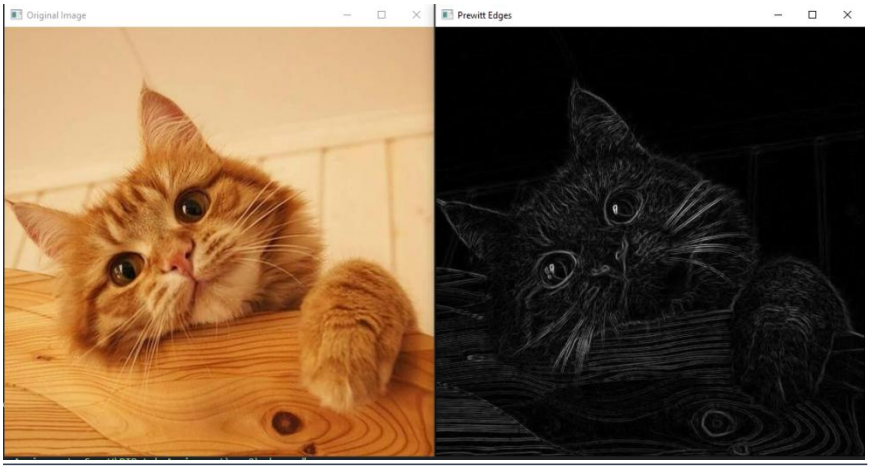
Background Theory: The Prewitt operator in image processing is a technique for edge detection. It calculates the gradient of an image by convolving it with 3x3 convolution kernels to highlight changes in intensity along the horizontal and vertical directions. The resulting gradients are used to compute the magnitude and direction of edges.

Python Coder:

```
import cv2
import numpy as np
def apply_prewitt_operator(image):
    # Convert the image to grayscale if it's a colored image
    if len(image.shape) == 3:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Get the dimensions of the image
    height, width = image.shape
    # Create empty matrices for horizontal and vertical gradients
    prewitt_x = np.zeros_like(image, dtype=np.float32)
    prewitt_y = np.zeros_like(image, dtype=np.float32)
    # Define Prewitt kernels for horizontal and vertical edges
    kernel_x = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])
    kernel_y = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])
    # Apply convolution using the Prewitt kernels
    for i in range(1, height - 1):
        for j in range(1, width - 1):
            prewitt_x[i, j] = np.sum(image[i-1:i+2, j-1:j+2] * kernel_x)
            prewitt_y[i, j] = np.sum(image[i-1:i+2, j-1:j+2] * kernel_y)
    # Calculate the gradient magnitude
    gradient_magnitude = np.sqrt(prewitt_x**2 + prewitt_y**2)
    # Normalize the gradient magnitude to the range [0, 255]
    gradient_magnitude_normalized = cv2.normalize(gradient_magnitude, None, 0, 255,
cv2.NORM_MINMAX, dtype=cv2.CV_8U)
    return gradient_magnitude_normalized
image = cv2.imread("ass9/cat.jpg")
# Apply the Prewitt operator for edge detection
```

```
prewitt_edges = apply_prewitt_operator(image)
# Display the original and Prewitt edges side by side
cv2.imshow("Original Image", image)
cv2.imshow("Prewitt Edges", prewitt_edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Input Image vs Output Image



Conclusion/Observation: This program finds the edge in the image using Prewitt Operator. From the output, we observe

- Prewitt operator effectively highlights edges with high gradient magnitudes.

- It is sensitive to edge orientations, enhancing horizontal and vertical edges differently.

Q10) Write a program to segment an image using the threshold method.

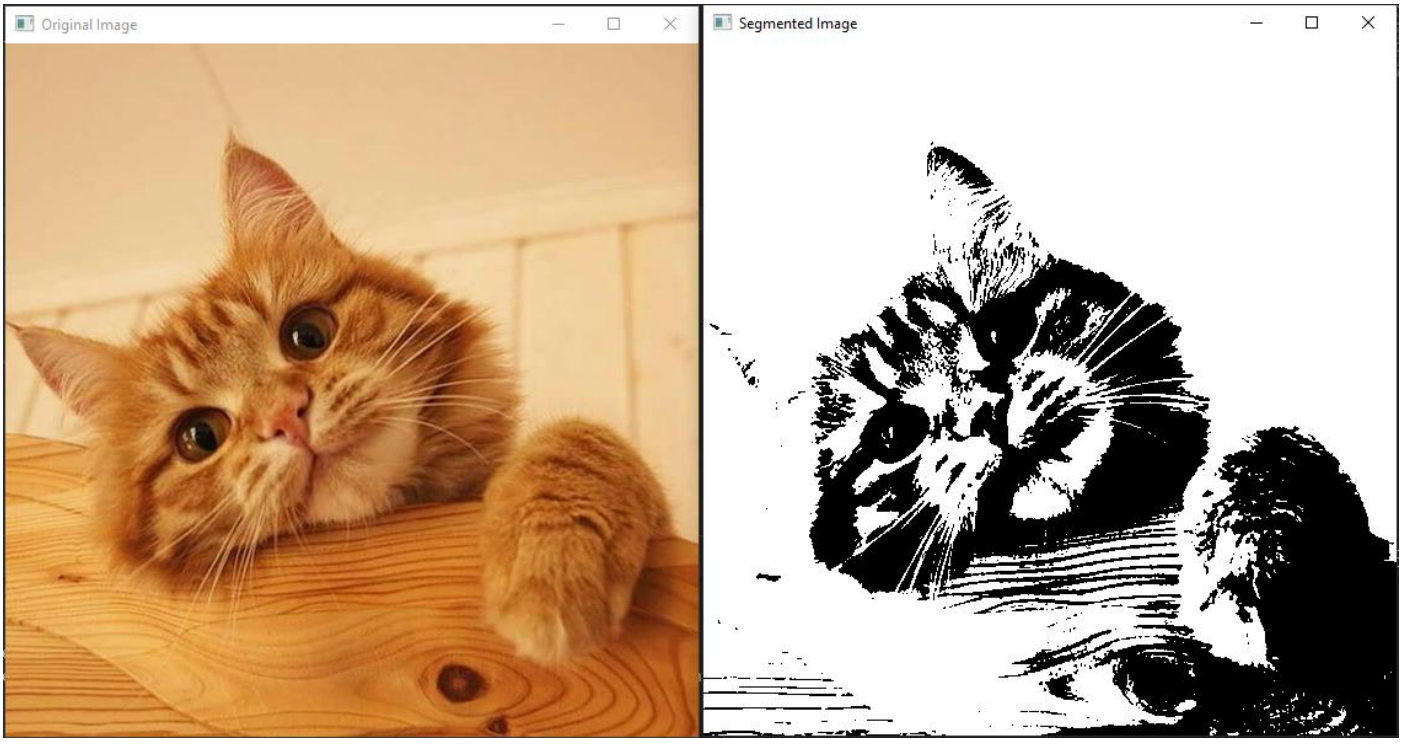
Problem Statement: Program to segment an image using threshold method and displaying the input and output image.

Background Theory: Threshold segmentation involves dividing an image based on pixel intensity values. A chosen threshold separates pixels into foreground (below threshold) and background (above threshold). This simple technique is used for image binarization, where pixels are classified into two distinct regions, aiding in object detection and simplifying further image analysis

Python Code:

```
import cv2
import numpy as np
def segment_image_manually(image_path, threshold_value):
    original_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    # Get the dimensions of the image
    height, width = original_image.shape
    # Create an empty matrix for the segmented image
    segmented_image = np.zeros((height, width), dtype=np.uint8)
    for i in range(height):
        for j in range(width):
            # Apply manual thresholding
            if original_image[i, j] > threshold_value:
                segmented_image[i, j] = 255
            else:
                segmented_image[i, j] = 0
    return segmented_image
image_path = "ass10/cat.jpg"
# Specify the threshold value for manual segmentation (adjust as needed)
threshold_value = 127
# Perform manual image segmentation using the threshold method
segmented_result = segment_image_manually(image_path, threshold_value)
# Display the original and segmented images
cv2.imshow("Original Image", cv2.imread(image_path))
cv2.imshow("Segmented Image (Manual)", segmented_result)
cv2.waitKey(0)
# Close windows
cv2.destroyAllWindows()
```

Input image vs Output image:



Conclusion/Observation: This program successfully segments the image. From the output, we can observe:

- Segmentation results in a binary image, with black and white pixels.

- Clear distinction between foreground and background based on threshold.