+

SAHYADRI
COLLEGE OF ENGINEERING & MANAGEMENT
An Autonomous Institution
MANGALURU

Department of Computer Science and Engineering

(Artificial Intelligence and Machine Learning)

DIGITAL IMAGE PROCESSING (21AI71)

LAB RECORD

Submitted by,

Yagnik S Ram

4SF21CI057

Department of Computer Science and Engineering

(Artificial Intelligence and Machine Learning)

Marks

Head of The Department

Signature of the
Faculty-Incharge of the batch

INDEX – LABORATORY

INDEX - ASSIGNMENT

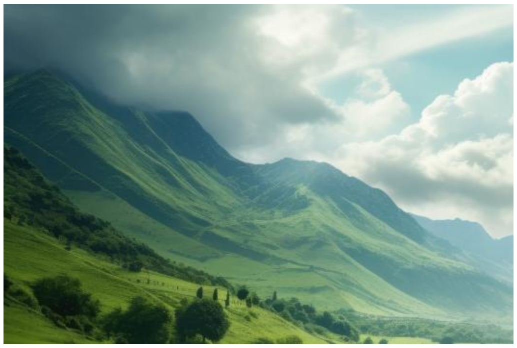
LAB PROGRAMS

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

```
# pip install opencv-python
import cv2

Image_path = 'LAB_1/Miguel.jpeg'
image = cv2.imread(image_path)
if image is None:
        print("Error: Unable to load image.")
exit()

print(image.shape)
height, width, _ = image.shape

center_y, center_x = height // 2, width // 2

top_left = image[0:center_y, 0:center_x]
top_right = image[0:center_y, center_x:]
bottom_left = image[center_y:, 0:center_x]
bottom_right = image[center_y:, center_x:]

cv2.imshow('Top Left', top_left)
cv2.imshow('Top Right', top_right)
cv2.imshow('Bottom Left', bottom_left)
cv2.imshow('Bottom Right', bottom_right)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT IMAGE

OUTPUT



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

```
import cv2

# Function to apply Gaussian Filter
def apply_gaussian_filter(image, kernel_size=5, sigma=1.0):
# Apply Gaussian Blur
gaussian_filtered = cv2.GaussianBlur(image, (kernel_size, kernel_size), sigma)
 return gaussian_filtered

# Function to apply Median Filter
def apply_median_filter(image, kernel_size=13):
# Apply Median Filter
median_filtered = cv2.medianBlur(image, kernel_size)
return median_filtered

# Main function
def main():
# Hard-coded image path
image_path = 'LAB_2/flower.jpg'  # Change this to your image path

# Read the image
image = cv2.imread(image_path)

if image is None:
        print("ERROR! Could not read the image.")
return

# Resize the image to a smaller size (e.g., 40% of the original)
new_width = int(image.shape[1] * 0.4)
new_height = int(image.shape[0] * 0.4)
resized_image = cv2.resize(image, (new_width, new_height))

# Apply Gaussian Filter
gaussian_result = apply_gaussian_filter(resized_image, kernel_size=5, sigma=1.0)

# Apply Median Filter
median_result = apply_median_filter(resized_image, kernel_size=5)

# Display the results
cv2.imshow('Original Image (Resized)', resized_image)
cv2.imshow('Gaussian Filtered Image', gaussian_result)
cv2.imshow('Median Filtered Image', median_result)
```

```
# Wait until a key is pressed, then close all windows
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    if __name__ == "__main__":
        main()
```

OUTPUT

Original Image

Gaussian Filtered

Median Filtered

3. Demonstrate enhancing and segmenting low contrast 2D images

```
import cv2
import numpy as np

# Load the image
image = cv2.imread('LAB_3\low_contrast_image.png', cv2.IMREAD_GRAYSCALE)

# Enhance the image using CLAHE
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
enhanced_image = clahe.apply(image)

# Segment the image using Otsu's thresholding
_, segmented_image = cv2.threshold(enhanced_image, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

# Display the results
cv2.imshow('Original Image', image)
cv2.imshow('Enhanced Image', enhanced_image)
cv2.imshow('Segmented Image', segmented_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT

Original Image

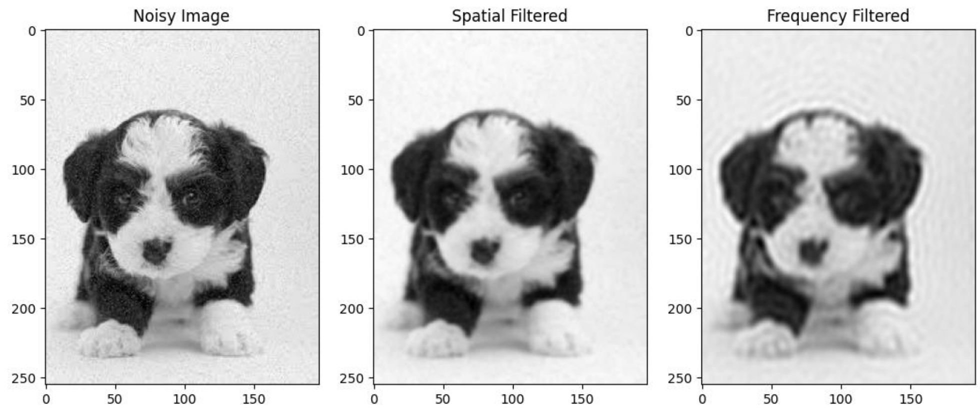
Enhanced Image

Segmented Image

4. Demonstrate image restoration using spatial or frequency domain

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

# Load the image
image = cv2.imread('LAB_4/noisy_dog.jpg', 0)

# Spatial Domain Example: Gaussian Filter
spatial_filtered = cv2.GaussianBlur(image, (5, 5), 0)

# Frequency Domain Example
# Step 1: Fourier Transform
f = np.fft.fft2(image)
fshift = np.fft.fftshift(f)

# Step 2: Create a mask (low-pass filter)
rows, cols = image.shape
crow, ccol = rows // 2, cols // 2
mask = np.zeros((rows, cols), np.uint8)
r = 30  # Radius of the mask
cv2.circle(mask, (ccol, crow), r, 1, thickness=-1)

# Step 3: Apply the mask
fshift_filtered = fshift * mask

# Step 4: Inverse Fourier Transform
f_ishift = np.fft.ifftshift(fshift_filtered)
image_restored = np.fft.ifft2(f_ishift)
image_restored = np.abs(image_restored)

# Display the results
plt.figure(figsize=(12, 6))
plt.subplot(1, 3, 1), plt.imshow(image, cmap='gray'), plt.title('Noisy Image')
plt.subplot(1, 3, 2), plt.imshow(spatial_filtered, cmap='gray'), plt.title('Spatial Filtered')
plt.subplot(1, 3, 3), plt.imshow(image_restored, cmap='gray'), plt.title('Frequency Filtered')
plt.show()
```

OUTPUT

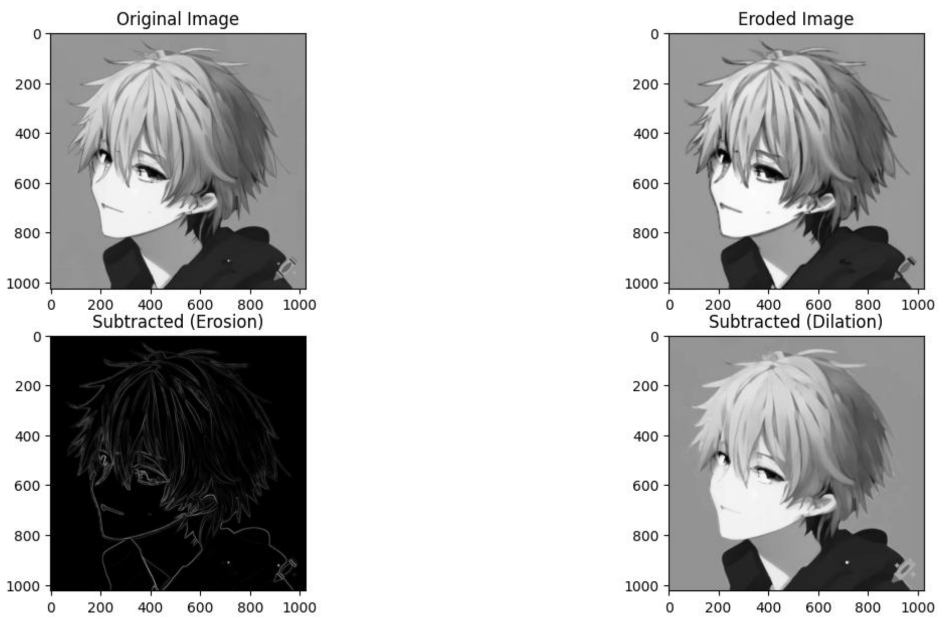


5. Read an image, first apply erosion to the image and then subtract the result from the original. Demonstrate the difference in the edge image if you use dilation instead of erosion.

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

# Read the image
image = cv2.imread('LAB_5/anime.jpg', cv2.IMREAD_GRAYSCALE)

# Define the kernel for erosion and dilation
kernel = np.ones((5, 5), np.uint8)

# Apply erosion
eroded_image = cv2.erode(image, kernel, iterations=1)

# Subtract the eroded image from the original
subtracted_image = cv2.subtract(image, eroded_image)

# Apply dilation
dilated_image = cv2.dilate(image, kernel, iterations=1)

# Subtract the dilated image from the original

# Display the results
```

```
plt.figure(figsize=(10, 8))

plt.subplot(2, 2, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')

plt.subplot(2, 2, 2)
plt.title('Eroded Image')
plt.imshow(eroded_image, cmap='gray')

plt.subplot(2, 2, 3)
plt.title('Subtracted (Erosion)')
plt.imshow(subtracted_image, cmap='gray')

plt.subplot(2, 2, 4)
plt.title('Subtracted (Dilation)')
plt.imshow(dilated_image, cmap='gray')

plt.tight_layout()
```

OUTPUT



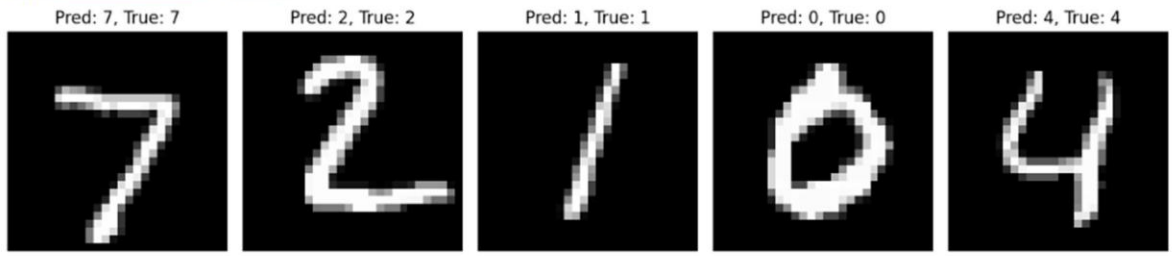
6. Implement image processing model using Computer Vision libraries

(TensorFlow, Keras)

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import layers, models from tensorflow.keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape((x_train.shape[0], 28, 28, 1)).astype('float32') / 255
x_test = x_test.reshape((x_test.shape[0], 28, 28, 1)).astype('float32') / 255 y_train =
tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

model = models.Sequential([
layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (3, 3), activation='relu'),
layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (3, 3), activation='relu'), layers.Flatten(),
layers.Dense(64, activation='relu'), layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=10, batch_size=128, validation_split=0.2)
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc:.4f}')

num_results = 5
predictions = model.predict(x_test)

plt.figure(figsize=(12, 6)) for i in range(num_results):
plt.subplot(1, num_results, i + 1) plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
plt.title(f"Pred: {np.argmax(predictions[i])}, True: {np.argmax(y_test[i])}") plt.axis('off')
plt.tight_layout() plt.show()
```

OUTPUT

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 14s 1us/step
Epoch 1/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 12s 25ms/step - accuracy: 0.7982 - loss: 0.6498 - val_accuracy: 0.9718 - val_loss: 0.0937
Epoch 2/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 9s 24ms/step - accuracy: 0.9748 - loss: 0.0808 - val_accuracy: 0.9818 - val_loss: 0.0588
Epoch 3/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 9s 25ms/step - accuracy: 0.9858 - loss: 0.0485 - val_accuracy: 0.9869 - val_loss: 0.0458
Epoch 4/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 9s 24ms/step - accuracy: 0.9899 - loss: 0.0327 - val_accuracy: 0.9857 - val_loss: 0.0508
Epoch 5/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 9s 24ms/step - accuracy: 0.9921 - loss: 0.0269 - val_accuracy: 0.9875 - val_loss: 0.0430
Epoch 6/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 9s 24ms/step - accuracy: 0.9932 - loss: 0.0202 - val_accuracy: 0.9883 - val_loss: 0.0397
Epoch 7/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 10s 25ms/step - accuracy: 0.9938 - loss: 0.0189 - val_accuracy: 0.9887 - val_loss: 0.0398
Epoch 8/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 10s 26ms/step - accuracy: 0.9954 - loss: 0.0140 - val_accuracy: 0.9878 - val_loss: 0.0418
Epoch 9/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 9s 25ms/step - accuracy: 0.9958 - loss: 0.0140 - val_accuracy: 0.9902 - val_loss: 0.0392
Epoch 10/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 9s 25ms/step - accuracy: 0.9965 - loss: 0.0115 - val_accuracy: 0.9899 - val_loss: 0.0406
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9868 - loss: 0.0390
Test accuracy: 0.9909
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step
```



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

```
import cv2
import numpy as np

# Load the image
image = cv2.imread('image.jpg')

# Get the dimensions of the image
(h, w) = image.shape[:2]

# Define the center of the image
center = (w // 2, h // 2)

# Rotation
angle = 45  # Rotate by 45 degrees
scale = 1.0  # No scaling during rotation
rotation_matrix = cv2.getRotationMatrix2D(center, angle, scale)
rotated_image = cv2.warpAffine(image, rotation_matrix, (w, h))

# Scaling
scale_x = 1.5  # Scale by 1.5 times along the x-axis
```

```
scale_y = 1.5  # Scale by 1.5 times along the y-axis
scaled_image = cv2.resize(image, None, fx=scale_x, fy=scale_y,

interpolation=cv2.INTER_LINEAR)

# Translation
tx = 100  # Translate by 100 pixels along the x-axis
ty = 50  # Translate by 50 pixels along the y-axis
translation_matrix = np.float32([[1, 0, tx], [0, 1, ty]])
translated_image = cv2.warpAffine(image, translation_matrix, (w, h))

# Display the results
cv2.imshow('Original Image', image)
cv2.imshow('Rotated Image', rotated_image)
cv2.imshow('Scaled Image', scaled_image)
cv2.imshow('Translated Image', translated_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT

Original Image

Rotated Image

Scaled Image

Translated Image

ASSIGNMENT PROBLEMS

1. Write a program to construct the histogram of an RGB image.

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

# Load an HD image with RGB layers (replace 'image.jpg' with
your image path)
image = cv2.imread('CLASS_PROBLEMS/Images/Flower.jpeg')
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  #
Convert to RGB

# Apply 2x2 average filter
kernel_2x2 = np.ones((2, 2), np.float32) / 4
filtered_2x2 = cv2.filter2D(image_rgb, -1, kernel_2x2)

# Apply 3x3 average filter
kernel_3x3 = np.ones((3, 3), np.float32) / 9
filtered_3x3 = cv2.filter2D(image_rgb, -1, kernel_3x3)

# Plot the original and filtered images
plt.figure(figsize=(15, 10))

plt.subplot(1, 3, 1)
```

```
plt.imshow(image_rgb)
plt.title("Original Image")
plt.axis("off")


plt.subplot(1, 3, 2)
plt.imshow(filtered_2x2)
plt.title("2x2 Average Filter")
plt.axis("off")


plt.subplot(1, 3, 3)
plt.imshow(filtered_3x3)
plt.title("3x3 Average Filter")
plt.axis("off")
plt.show()
```

OUTPUT

2. Write a code to apply linear, gamma, and logarithmic transformations to an image.

```
import cv2
import numpy as np

# Load the image
image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)

# Linear Transformation
alpha = 1.5  # Simple contrast control
beta = 50    # Simple brightness control
linear_transformed = cv2.convertScaleAbs(image, alpha=alpha, beta=beta)

# Gamma Transformation
gamma = 2.0
gamma_corrected = np.array(255 * (image / 255) ** gamma, dtype='uint8')

# Logarithmic Transformation
c = 255 / np.log(1 + np.max(image))
log_transformed = c * (np.log(image + 1))
log_transformed = np.array(log_transformed, dtype='uint8')

# Display the results
cv2.imshow('Original Image', image)
cv2.imshow('Linear Transformed Image', linear_transformed)
cv2.imshow('Gamma Corrected Image', gamma_corrected)
cv2.imshow('Logarithmic Transformed Image', log_transformed)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT

Original Image

Linear Transformed Image

Gamma Corrected image

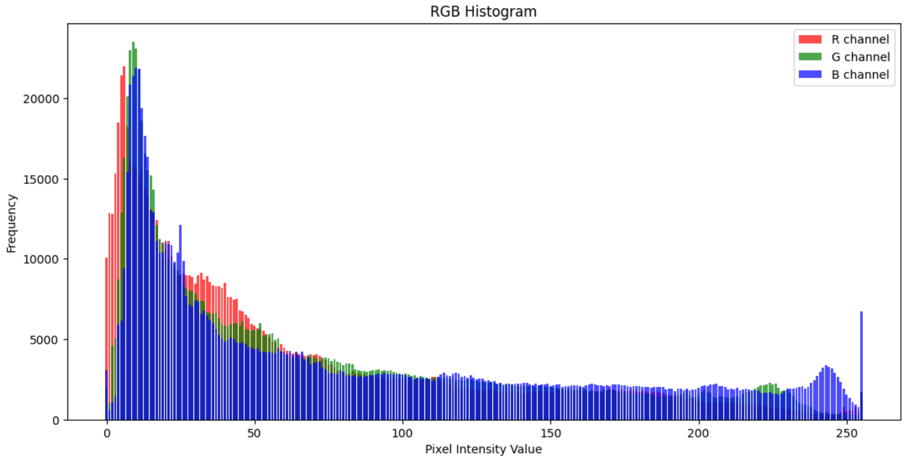
Logarithmic Transformed Image

3. Write a program to construct the histogram for a grayscale image.

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

# Load the image using OpenCV
image =
cv2.imread('CLASS_PROBLEMS/Im
ages/Flower.jpeg',
```

```
cv2.IMREAD_COLOR)

# Convert the image to RGB (OpenCV
loads images in BGR format by
default)
image_rgb = cv2.cvtColor(image,
cv2.COLOR_BGR2RGB)

# Create a figure for the bar plot
plt.figure(figsize=(10, 6))

# Initialize the number of bins (256 for
pixel values 0-255)
bins = 256

# Create a histogram for each color
channel and plot bars
colors = ('r', 'g', 'b')  # Red, Green,
Blue channels
for i, color in enumerate(colors):
    hist = cv2.calcHist([image_rgb], [i],
None, [bins], [0, 256])
    hist = hist.flatten()  # Flatten the
histogram array

    # Create bar plot for each channel
    plt.bar(np.arange(bins), hist,
color=color, alpha=0.7,
label=f'{color.upper()} channel')

# Add labels and title
plt.title('RGB Histogram')
plt.xlabel('Pixel Intensity Value')
plt.ylabel('Frequency')
plt.legend()

# Display the histogram
plt.show()
```

OUTPUT



4. Write Python code to convert a given image to a negative image.

```
import cv2

import numpy as np


# Load the image

image = cv2.imread('image.jpg')


# Convert the image to a negative image

negative_image = cv2.bitwise_not(image)


# Display the results

cv2.imshow('Original Image', image)

cv2.imshow('Negative Image', negative_image)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

OUTPUT


Original Image


Negative Image

5. Write a Python code to convert an RGB image to grayscale and display the R, G, and B components of the image.

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

def display_images(images, titles):
    plt.figure(figsize=(15, 5))
    for i, (image, title) in enumerate(zip(images, titles)):
        plt.subplot(1, len(images), i + 1)
        if len(image.shape) == 2:  # Grayscale image
            plt.imshow(image, cmap='gray')
        else:
            plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
        plt.title(title)
        plt.axis('off')

plt.show() image_path = "1.jpg"
image = cv2.imread(image_path)

if image is None:
    print("Error: Could not read the image. Please check the file path.")
else:
    B, G, R = cv2.split(image)


    R_colored = cv2.merge([np.zeros_like(B), np.zeros_like(G), R])  # Red channel
    G_colored = cv2.merge([np.zeros_like(B), G, np.zeros_like(R)])  # Green channel
```

```
    B_colored = cv2.merge([B, np.zeros_like(G), np.zeros_like(R)])  # Blue channel


    grayscale_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    display_images(
        [image, grayscale_image, R_colored, G_colored, B_colored],
        ["Original Image", "Grayscale Image", "Red Component", "Green Component", "Blue
Component"]
    )
```

OUTPUT

Original Image

Grayscale Image

Red Component

Green Component

Blue Component

6. Write a Python code to pixelate, blur, and add noise to an image.

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


# Load an HD image
image = cv2.imread('CLASS_PROBLEMS/Images/Sky.jpg')
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  #
Convert to RGB format
image_rgb = cv2.resize(image_rgb, (720, 1080))


# Function to add Gaussian noise
def add_gaussian_noise(image, mean=0, stddev=25):
    # Generate Gaussian noise
    noise = np.random.normal(mean, stddev, image.shape)
```

```
    noise = noise.reshape(image.shape)

    # Add noise to the image and clip values to maintain valid pixel
range
    noisy_image = image + noise
    noisy_image = np.clip(noisy_image, 0, 255)  # Keep pixel values
within [0, 255]

    return noisy_image.astype(np.uint8)

# Spoil the image by adding Gaussian noise
spoiled_image = add_gaussian_noise(image_rgb, mean=0,
stddev=30)
# Display the original and spoiled images
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(image_rgb)
plt.title("Original Image")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(spoiled_image)
plt.title("Spoiled Image with Gaussian Noise")
plt.axis("off")
plt.show()
```

OUTPUT

7. Write a program to enhance contrast of an image using histogram equlisation. Display the results with and without equalisation for comparision.

```
import cv2
import numpy as np

# Load the image
image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)

# Apply histogram equalization
equalized_image = cv2.equalizeHist(image)

# Display the results
cv2.imshow('Original Image', image)
cv2.imshow('Equalized Image', equalized_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT

Original Image

Equalized image