M.Sc. (Five Year Integrated) in Computer Science
(Artificial Intelligence & Data Science)

Fourth Semester

DIGITAL SIGNAL PROCESSING ASSIGNMENT

Submitted by

ASHFAQ HUSSAIN M S
DEPARTMENT OF COMPUTER SCIENCE

COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY (CUSAT)

KOCHI-682022

MARCH 2024

Contents

SIMPLE AVERAGING FILTERS

The moving average is the most common filter in DSP, mainly because it is the easiest digital filter to understand and use. In spite of its simplicity, the moving average filter is optimal for a common task: reducing random noise while retaining a sharp step response. This makes it the premier filter for time domain encoded signals. However, the moving average is the worst filter for frequency domain encoded signals, with little ability to separate one band of frequencies from another. The simple moving average filter averages recent values of the filter input for a given number of inputs.

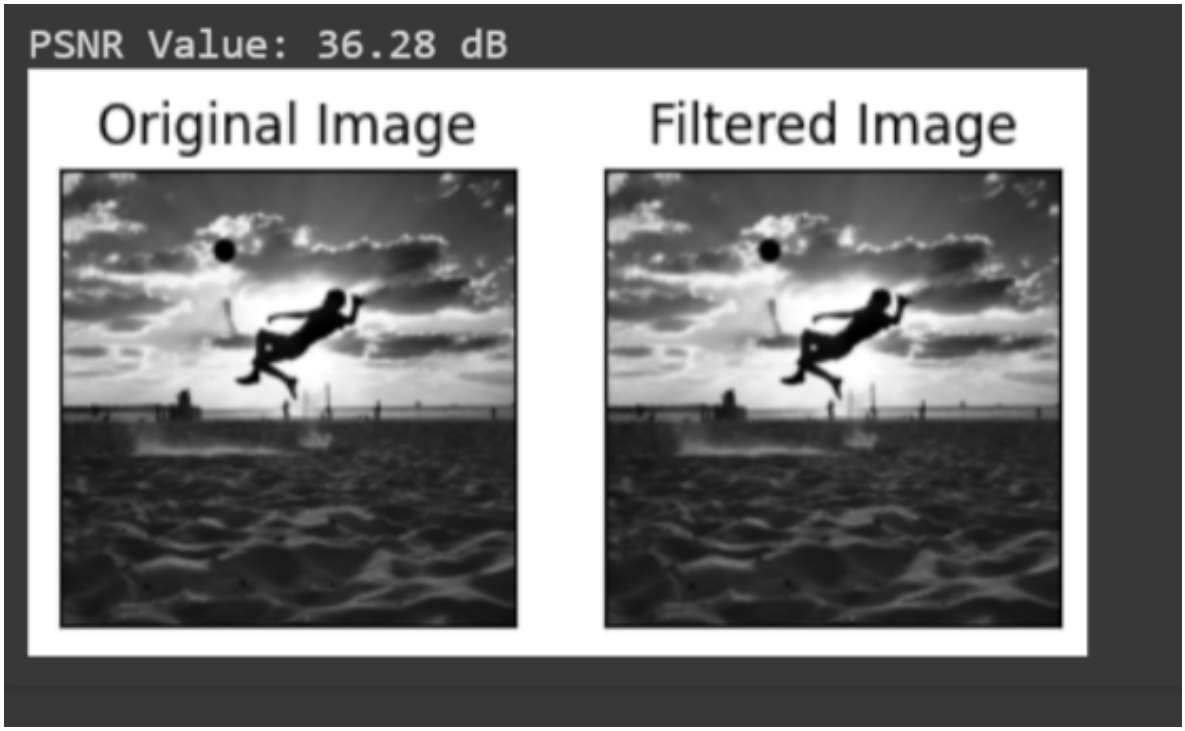
PROGRAM

```
#Simple Averaging Filter
import cv2
import numpy as np
import matplotlib.pyplot as plt

def apply_average_filter(image):
    # Define a 3x3 simple average filter
    kernel = np.ones((3, 3), np.float32) / 9

    # Apply the filter using OpenCV's filter2D function
    filtered_image = cv2.filter2D(image, -1, kernel)

    return filtered_image

def calculate_psnr(original_image, noisy_image, filtered_image):
    # Calculate mean squared error between original and filtered images
    mse_filtered = np.mean((original_image - filtered_image)**2)

    # Calculate PSNR
    psnr = 20 * np.log10(255 / np.sqrt(mse_filtered))

    return psnr

# Load the noisy image
image_path = '/content/probDS2.png'
noisy_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
```

```
# Check if the image is loaded successfully
if noisy_image is None:
    print(f"Error: Unable to load the image at path {image_path}")
else:
    # Apply the 3x3 simple average filter
    filtered_image = apply_average_filter(noisy_image)

    # Load the original image for PSNR calculation
    original_image_path = '/content/probDS2.png'
    original_image = cv2.imread(original_image_path, cv2.IMREAD_GRAYSCALE)

    # Check if the original image is loaded successfully
    if original_image is None:
        print(f"Error: Unable to load the original image at path
        {original_image_path}")
    else:
        # Calculate and print PSNR value
        psnr_value = calculate_psnr(original_image, noisy_image,
        filtered_image)
        print(f"PSNR Value: {psnr_value:.2f} dB")

        # Display the original, noisy, and filtered images
        plt.subplot(131), plt.imshow(original_image, cmap='gray')
        plt.title('Original Image'), plt.xticks([]), plt.yticks([])
        plt.subplot(132), plt.imshow(noisy_image, cmap='gray')
        plt.title('Noisy Image'), plt.xticks([]), plt.yticks([])
        plt.subplot(133), plt.imshow(filtered_image, cmap='gray')
        plt.title('Filtered Image'), plt.xticks([]), plt.yticks([])

        plt.show()
```

OUTPUT

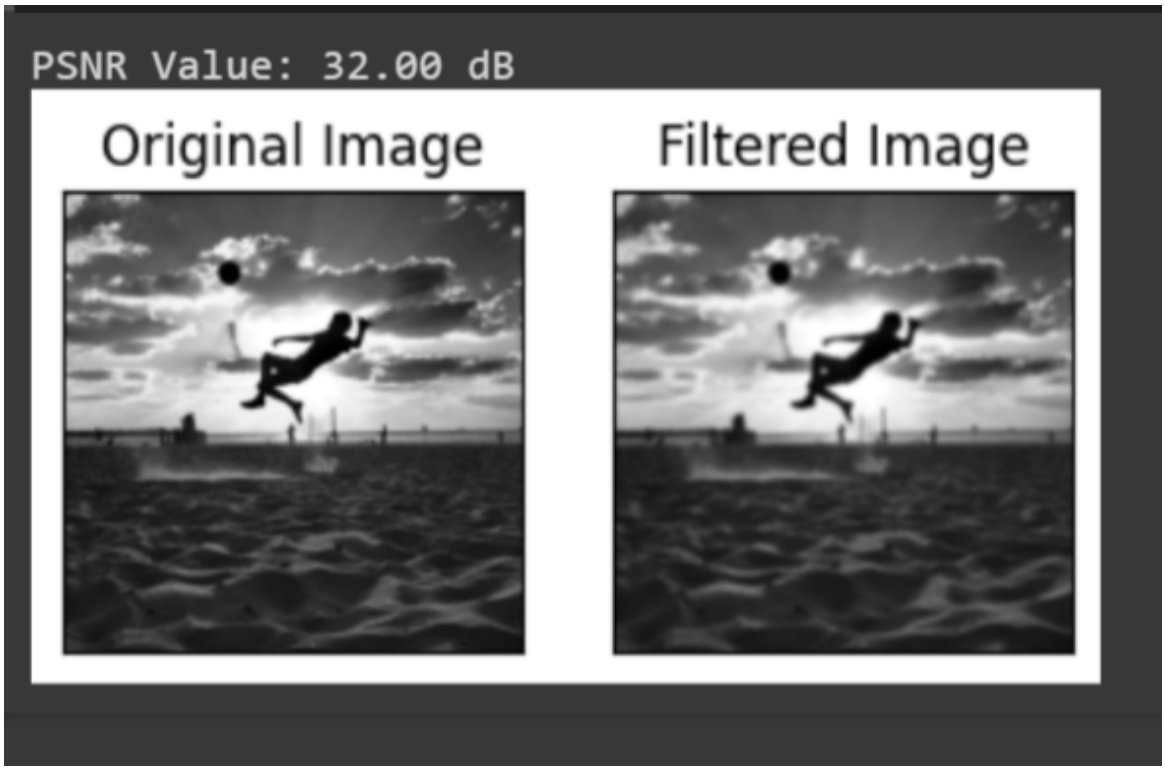
- 3×3 Average Filter



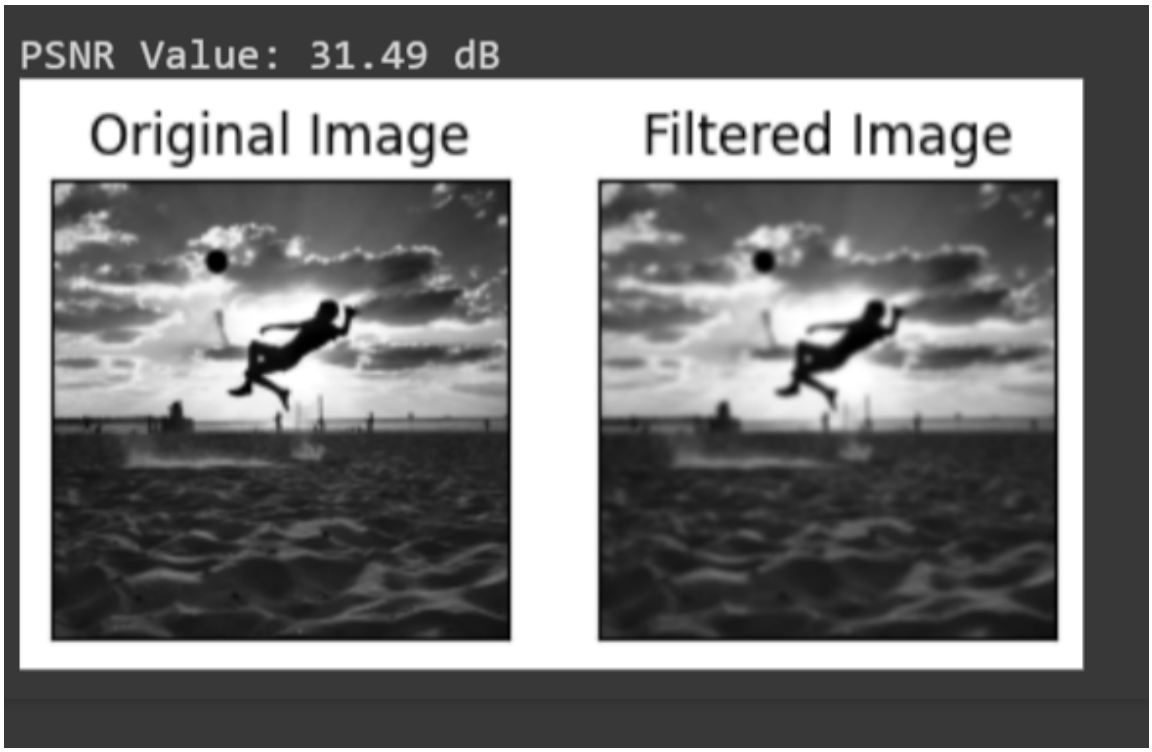
- 5×5 Average Filter

- 7×7 Average Filter

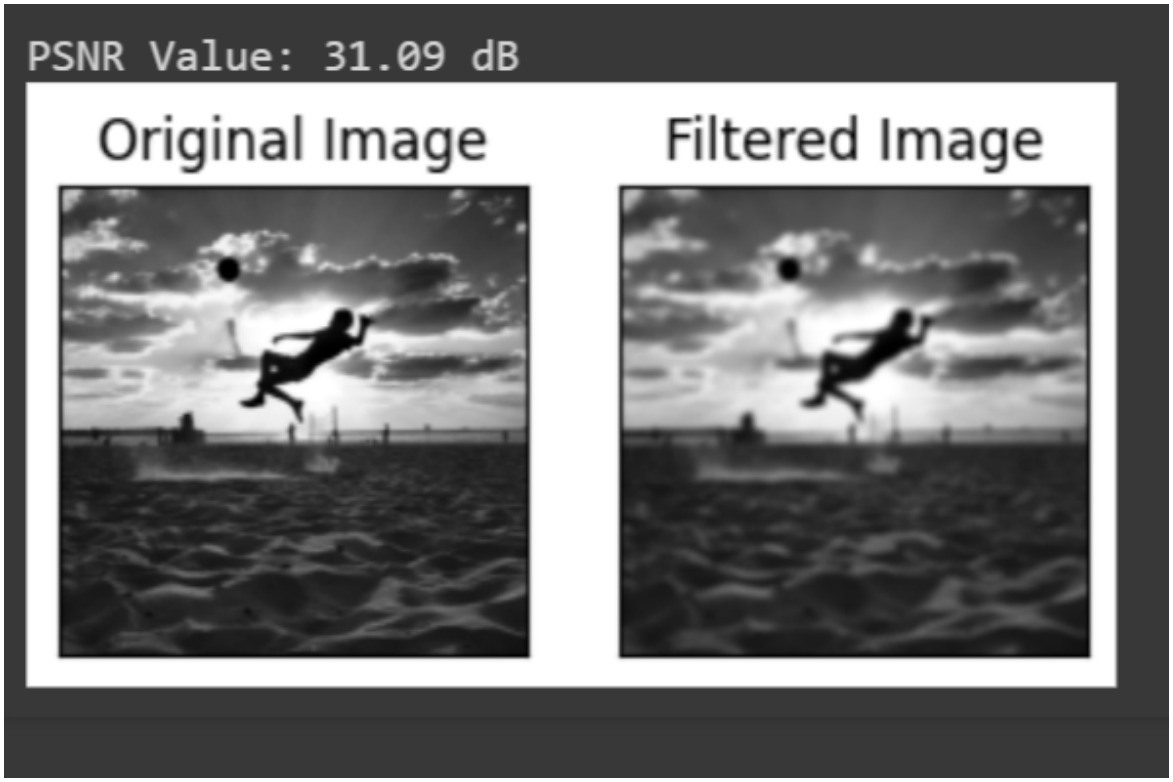
- 9 × 9 Average Filter



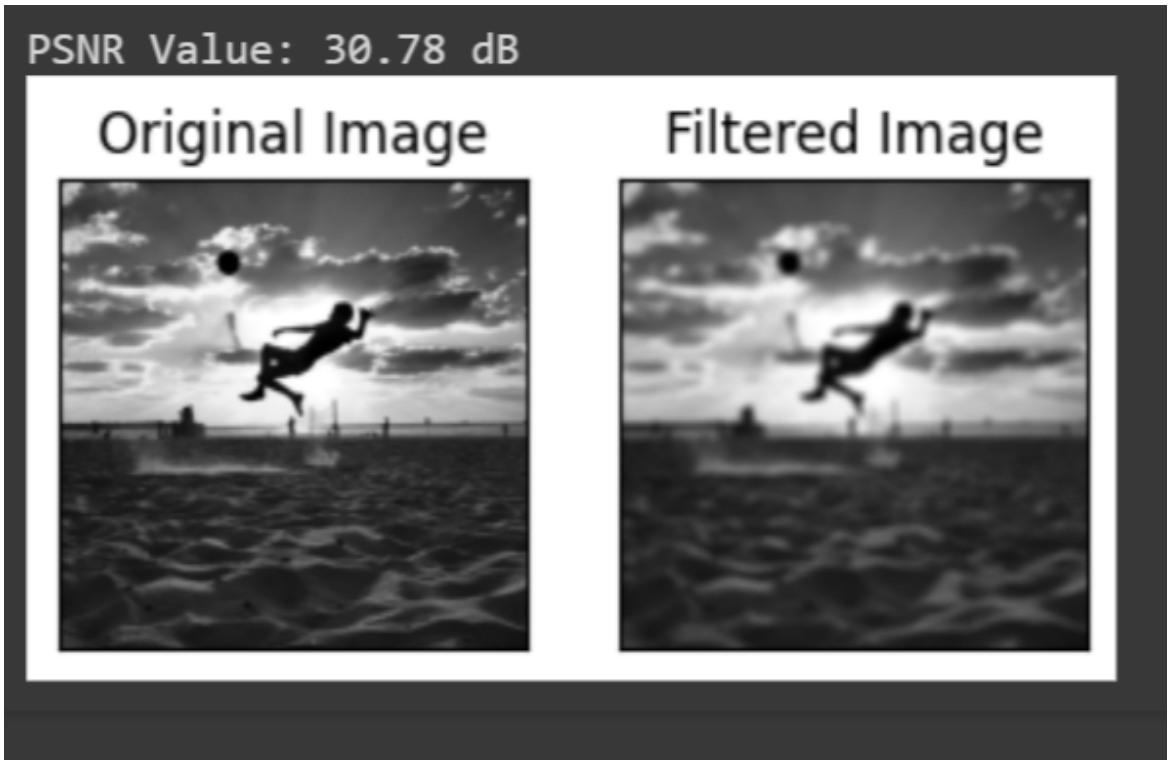
- 11 × 11 Average Filter



- 13 × 13 Average Filter



- 15 × 15 Average Filter



GAUSSIAN AVERAGING FILTERS

The Gaussian Smoothing Operator performs a weighted average of surrounding pixels based on the Gaussian distribution. It is used to remove Gaussian noise and is a realistic model of defocused lens. Sigma defines the amount of blurring.

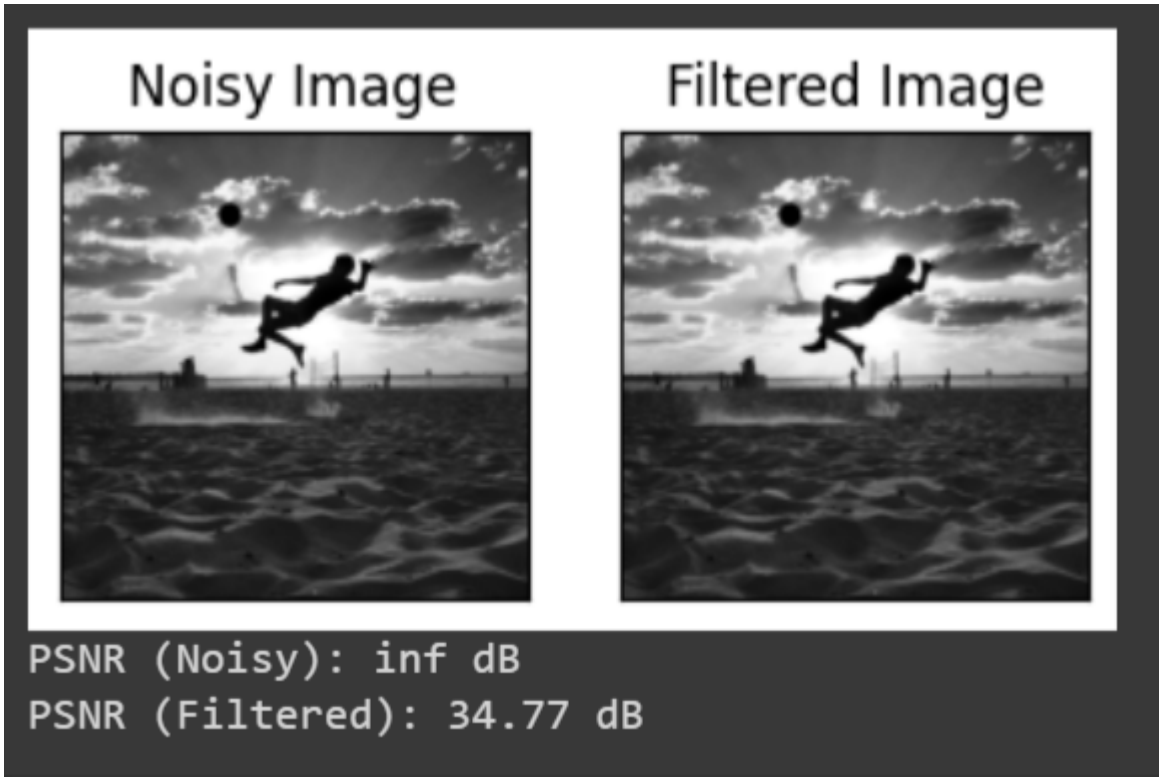
PROGRAM

```
#Gaussian Averaging Filter
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage.metrics import peak_signal_noise_ratio

def apply_gaussian_filter(image, kernel_size=(3, 3), sigma=1.0):
    # Apply a Gaussian filter using OpenCV's GaussianBlur function
    filtered_image = cv2.GaussianBlur(image, kernel_size, sigma)

    return filtered_image

def calculate_psnr(original, noisy, filtered):
    # Calculate PSNR for both noisy and filtered images
    psnr_noisy = peak_signal_noise_ratio(original, noisy)
    psnr_filtered = peak_signal_noise_ratio(original, filtered)

    return psnr_noisy, psnr_filtered

# Load the noisy image
image_path = '/content/DSP-2.jpg'
noisy_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Check if the image is loaded successfully
if noisy_image is None:
    print(f"Error: Unable to load the image at path {image_path}")
else:
    # Apply the Gaussian filter
    filtered_image = apply_gaussian_filter(noisy_image)
```

```
# Calculate PSNR values
psnr_noisy, psnr_filtered = calculate_psnr(noisy_image, noisy_image,
filtered_image)

# Display the original, noisy, and filtered images
plt.subplot(131), plt.imshow(noisy_image, cmap='gray')
plt.title('Noisy Image'), plt.xticks([]), plt.yticks([])
plt.subplot(132), plt.imshow(filtered_image, cmap='gray')
plt.title('Filtered Image'), plt.xticks([]), plt.yticks([])
plt.subplot(133), plt.imshow(noisy_image - filtered_image,
cmap='gray')
plt.title('Difference (Noisy - Filtered)'), plt.xticks([]),
plt.yticks([])

plt.show()

# Print PSNR values
print(f"PSNR (Noisy): {psnr_noisy:.2f} dB")
print(f"PSNR (Filtered): {psnr_filtered:.2f} dB")
```

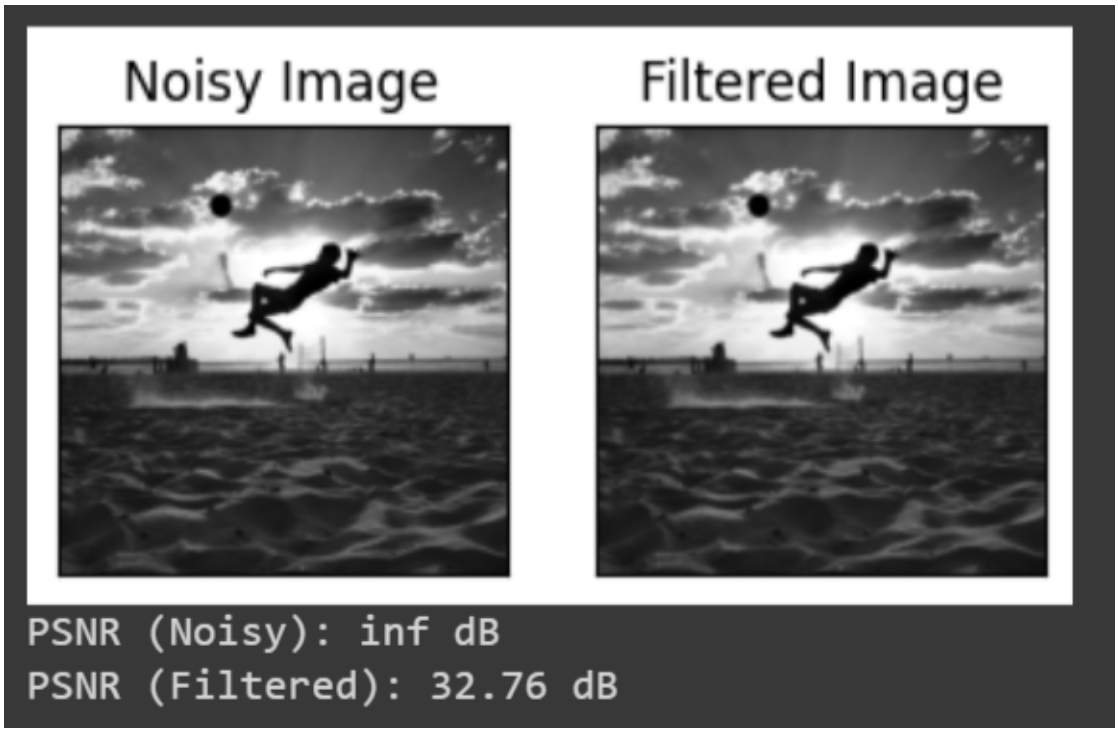
OUTPUT

- 3 × 3 Gaussian Filter



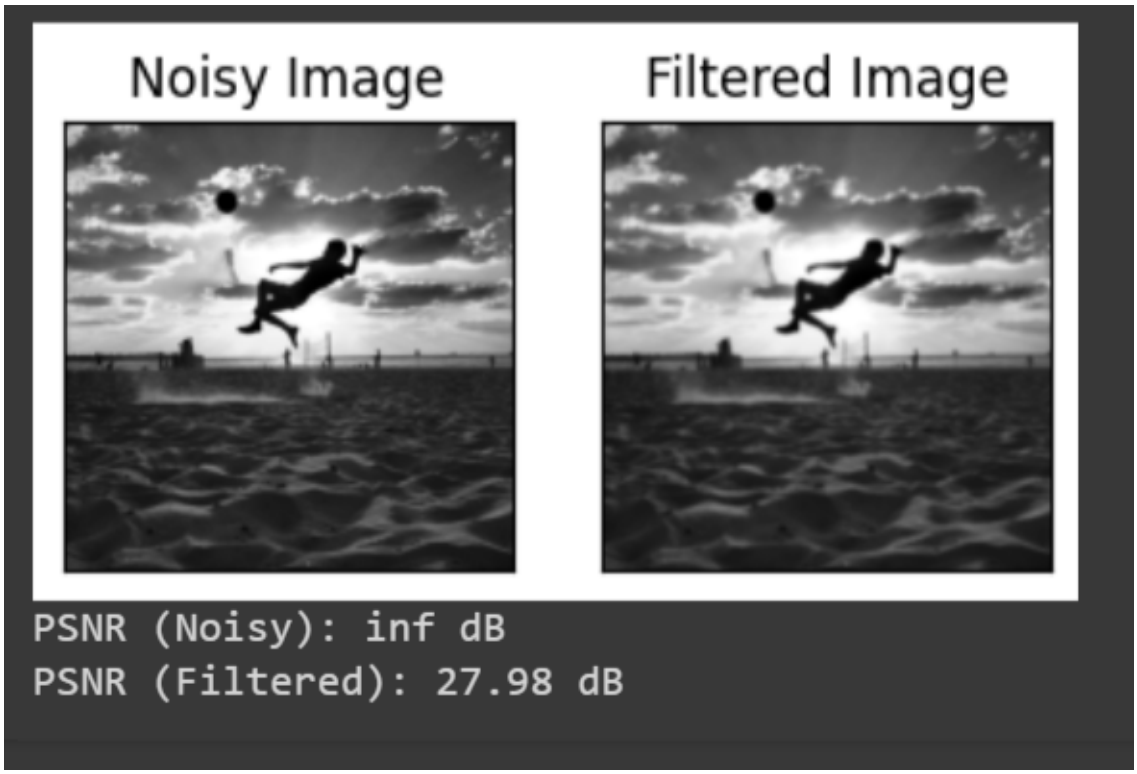
- 5 × 5 Gaussian Filter

- 7×7 Gaussian Filter



- 9×9 Gaussian Filter

- 11 × 11 Gaussian Filter



- 13 × 13 Gaussian Filter

- 15 × 15 Gaussian Filter