EXPERIMENT NO. 3

Title:

To perform Image Filtering (Smoothing, Sharpening).

Objective:

- To understand and implement image filtering techniques such as smoothing and sharpening.
- To apply various filtering techniques to enhance image quality.
- To observe the effects of different filters on an image.

.

Brief Theory:

Image Filtering:

Image filtering is a technique used to process an image by modifying its pixel values based on neighboring pixels. Two common filtering techniques are:

1. Smoothing (Blurring):

   - Reduces noise and details in an image.
   - Useful for preprocessing tasks like edge detection.
   - Common smoothing techniques include:
     1. **Averaging Filter**: Uses the mean of neighboring pixels.
     2. **Gaussian Filter**: Uses a weighted mean for better smoothness.
     3. **Median Filter**: Replaces a pixel with the median of neighboring values (useful for salt-and-pepper noise removal).
     4. **Bilateral Filter**: Preserves edges while reducing noise.

2. Sharpening:

   - Enhances the edges and details of an image.
   - Common sharpening techniques include:
     - **Laplacian Filter**: Highlights edges by calculating second-order derivatives.
     - **Unsharp Masking**: Enhances details by subtracting a blurred version of the image from the original.

Steps: -

Step 1: Import Necessary Libraries

First, we need to import the libraries required for Image Processing Essentials on Google Colab.

Code: -

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
from google.colab import files
```

Step 2: Read the Image

We will read the sample image from the local directory.

Code: -

```
img = cv2.imread('sample.jpg')
```

imread() reads the image from source.

Step 3: Convert the Sample image to RGB image for correct color display.

Code: -

```
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

Step 4: Apply Smoothing (Blurring) Filters.

1. **Average (Mean) Blur**

Code: -

```
avg_blur = cv2.blur(img, (5, 5))
```

- `cv2.blur(img, (5,5))`: Averages pixel values over a **5×5** kernel (window).
- This results in a **smooth and softened** image.

2. **Gaussian Blur**

Code: -

```
gaussian_blur = cv2.GaussianBlur(img, (5, 5), 0)
```

- `cv2.GaussianBlur(img, (5,5), 0)`: Uses a Gaussian filter instead of a simple average.
- Produces a **more natural blur** compared to simple averaging.

3. **Median Blur**

Code: -

```
median_blur = cv2.medianBlur(img, 5)
```

- `cv2.medianBlur(img, 5)`: Replaces each pixel with the median value in a **5×5** window.
- **Great for removing salt-and-pepper noise** while preserving edges.

4. **Bilateral Blur**

Code: -

```
bilateral_blur = cv2.bilateralFilter(img, 9, 75, 75)
```

- `cv2.bilateralFilter(img, 9, 75, 75):`
- **Preserves edges** while reducing noise.
    - 9: Kernel size.
    - 75: Intensity difference parameter.
    - 75: Space difference parameter.

Step 5: Apply Sharpening Filter

Code: -

```
sharpening_kernel = np.array([[-1, -1, -1],
                              [-1, 9, -1],
                              [-1, -1, -1]])
sharpened = cv2.filter2D(img, -1, sharpening_kernel)
```

- Creates a **sharpening kernel** (3×3 matrix).
- `cv2.filter2D(img, -1, sharpening_kernel)`: Convolves the kernel with the image.
- Enhances edges by highlighting high-frequency components.

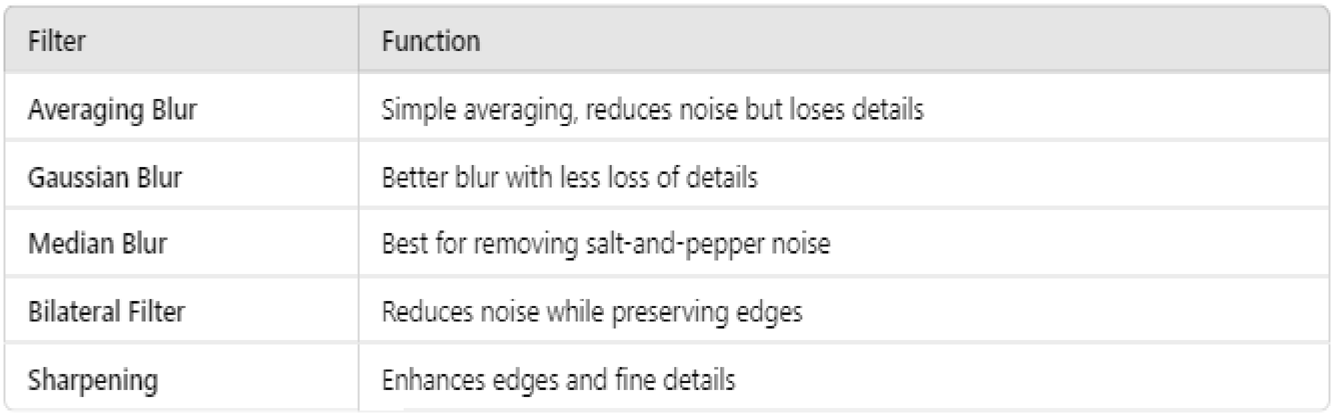
Step 6: Display the Images

Code: -

```
fig, ax = plt.subplots(2, 3, figsize=(15, 10))

ax[0, 0].imshow(img)
ax[0, 0].set_title("Original Image")
ax[0, 0].axis("off")

ax[0, 1].imshow(avg_blur)
ax[0, 1].set_title("Averaging Blur")
ax[0, 1].axis("off")

ax[0, 2].imshow(gaussian_blur)
ax[0, 2].set_title("Gaussian Blur")
ax[0, 2].axis("off")

ax[1, 0].imshow(median_blur)
ax[1, 0].set_title("Median Blur")
ax[1, 0].axis("off")

ax[1, 1].imshow(bilateral_blur)
ax[1, 1].set_title("Bilateral Filtering")
ax[1, 1].axis("off")

ax[1, 2].imshow(sharpened)
ax[1, 2].set_title("Sharpened Image")
ax[1, 2].axis("off")

plt.tight_layout()
plt.show()
```

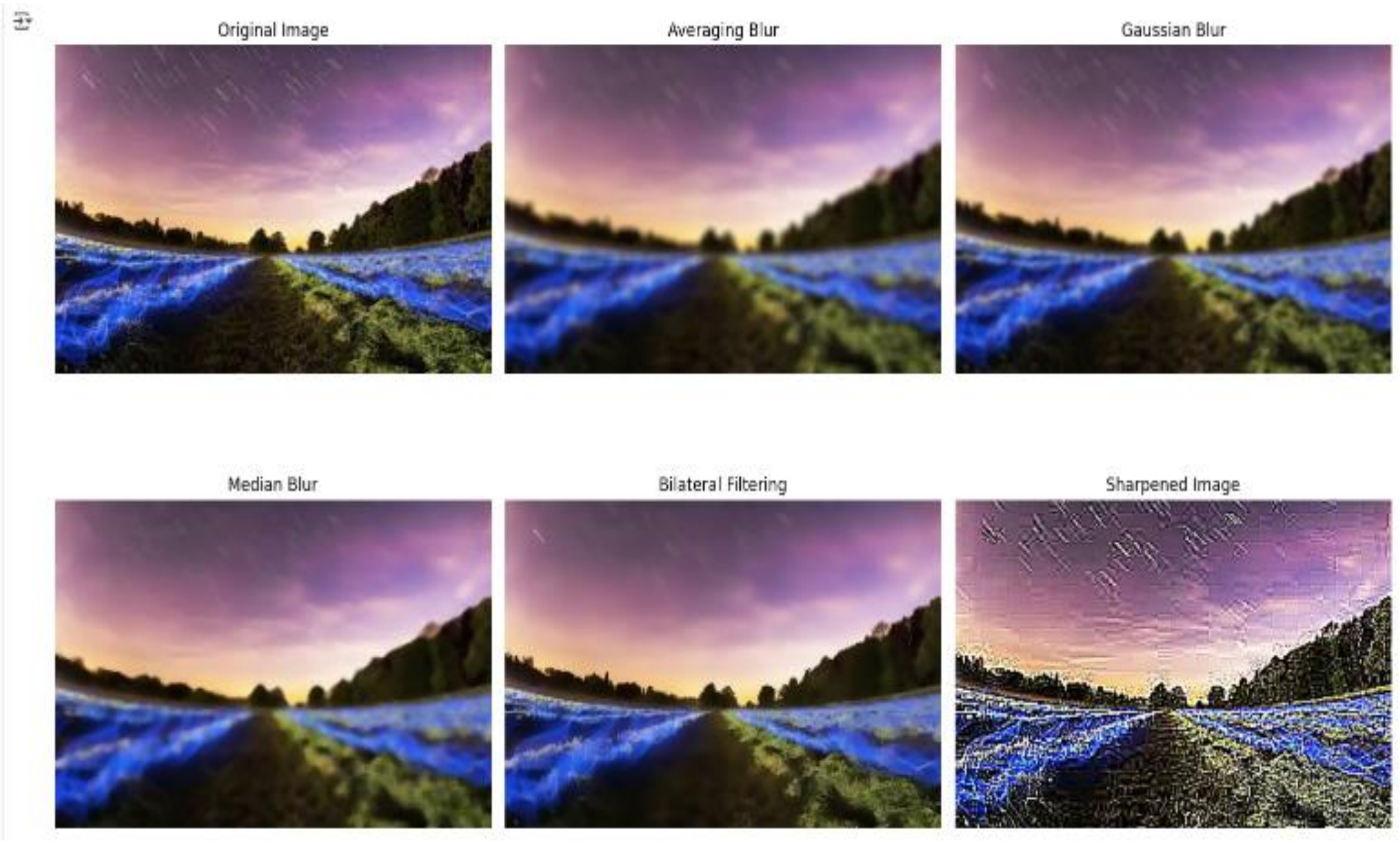
- Displays each filtered image with a corresponding title.

➢ **Summary of Filters**



Filter	Function
Averaging Blur	Simple averaging, reduces noise but loses details
Gaussian Blur	Better blur with less loss of details
Median Blur	Best for removing salt-and-pepper noise
Bilateral Filter	Reduces noise while preserving edges
Sharpening	Enhances edges and fine details

Output:



Conclusion:

This experiment demonstrates various image filtering techniques:

- **Smoothing filters** (blurring) remove noise.
- **Sharpening filters** enhance details.
- Each method has specific use cases in image processing.

Practice Questions:

1. Apply a bilateral filter to preserve edges while smoothing an image.
2. Create a custom kernel to detect horizontal edges in an image.
3. Compare the effects of different kernel sizes in Gaussian filtering.
4. Implement image sharpening using the Unsharp Masking technique.
5. Perform adaptive thresholding after applying smoothing filters.

Expected Oral Questions

1. What is the difference between averaging and Gaussian filtering?
2. How does a median filter remove noise better than an averaging filter?
3. What is the purpose of sharpening in image processing?
4. How does a convolutional kernel work in filtering?
5. What is the difference between smoothing and edge detection?
6. Why does Gaussian blur use a weighted average instead of a simple mean?

FAQs in Interviews

1. **What are the different types of image filtering techniques?**
   - Smoothing (Averaging, Gaussian, Median), Sharpening (Laplacian, Unsharp Masking), Edge Detection (Sobel, Canny).
2. **What is the role of convolutional kernels in filtering?**
   - They help apply transformations by calculating weighted sums of neighboring pixels.
3. **How does a median filter handle salt-and-pepper noise?**
   - It replaces a pixel with the median value of its neighbors, reducing noise efficiently.
4. **Why is Gaussian blur preferred over an averaging filter?**
   - It provides smoother results by giving more weight to central pixels.
5. **What is the significance of the Laplacian filter?**
   - It enhances edges by calculating second-order derivatives.
6. **How can image filtering be applied in real-world applications?**
   - Used in noise reduction, feature enhancement, object detection, and medical imaging.