Problem 1 – What happened here (20 points):

Problem1.py is our Python script containing all the functions.

This Python script applies various spatial operations and filters to a base grayscale image to recreate and analyze modified versions of the image. Each operation, such as blurring, sharpening, edge detection, and noise reduction, is applied to simulate different effects. The script processes each image by comparing it to the original, calculates the differences using Mean Squared Error (MSE), and saves the results for further analysis.

The next pages contain detailed explanations for each processed image, including:

1. The operation identified.
2. The reasoning behind the identification.
3. The effects of the operation on the image.
4. Bonus details, such as filter parameters and MSE values.

Image 1:

Operation Identified: Horizontal Blur

Reasoning:

- A horizontal averaging filter with a kernel size of (1, 10000) was applied to the image.
- The image appears smoothed along the horizontal axis, with pixel values averaged across rows. This operation reduces high-frequency noise and details horizontally, creating a uniform and smoothed appearance.
- The lines in the image appear smeared horizontally, representing an average of the pixel intensities.

Effects:

- This operation reduces noise and details horizontally while preserving the overall brightness.
- The transformation emphasizes the overall intensity gradients along the horizontal direction.

Bonus Section:

- Kernel/Filter Used: (1, 10000) averaging kernel.
- MSE Value: 0.15411885935082645.

Result:

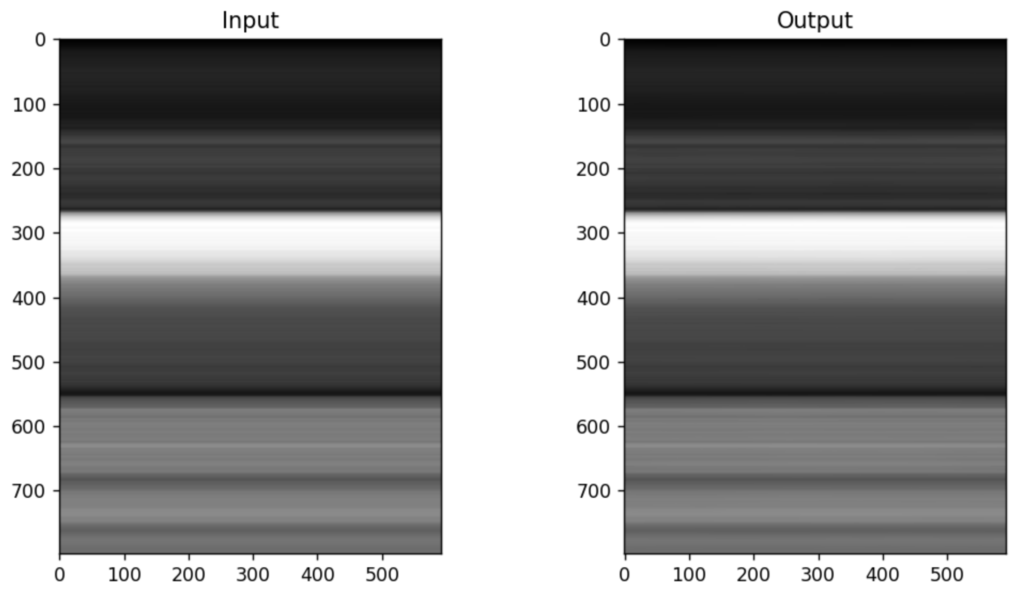


Image 2:

Operation Identified: Gaussian Blur

Reasoning:

- A Gaussian blur filter with a kernel size of (13, 13) and a standard deviation of 13.0 was applied to the image.
- The image appears softened with reduced noise and smoother transitions between intensity levels.
- Sharp edges and fine details are less emphasized, while the overall structure of the image is preserved.

Effects:

- This operation reduces noise and smooths sharp edges.
- It creates a uniformly softened appearance by distributing pixel intensities based on their proximity to the center of the kernel.

Bonus Section:

- **Kernel/Filter Used**: Gaussian kernel with size (13, 13) and standard deviation 13.0.
- **MSE Value**: 11.984604670459934.

Result:

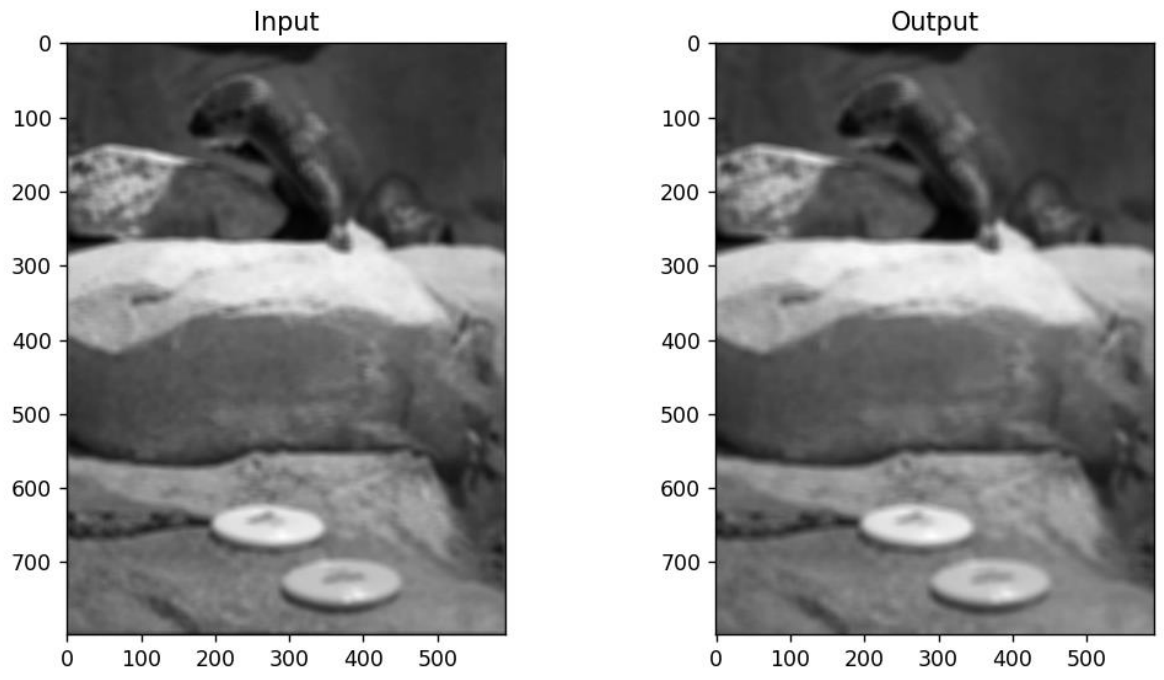


Image 3:

Operation Identified: Median Filter

Reasoning:

- A median filter with a kernel size of (11, 11) was applied to the image.
- The filter replaces each pixel with the median value of its neighboring pixels, effectively removing salt-and-pepper noise.
- The image appears cleaner, with noise reduced while edges and sharp details are preserved.

Effects:

- This operation reduces noise, particularly salt-and-pepper noise, while maintaining sharp transitions between objects and their surroundings.
- It strikes a balance between noise reduction and edge preservation.

Bonus Section:

- **Kernel/Filter Used**: Median filter with kernel size (11, 11).
- **MSE Value**: 0.36766112917428706.

Result:

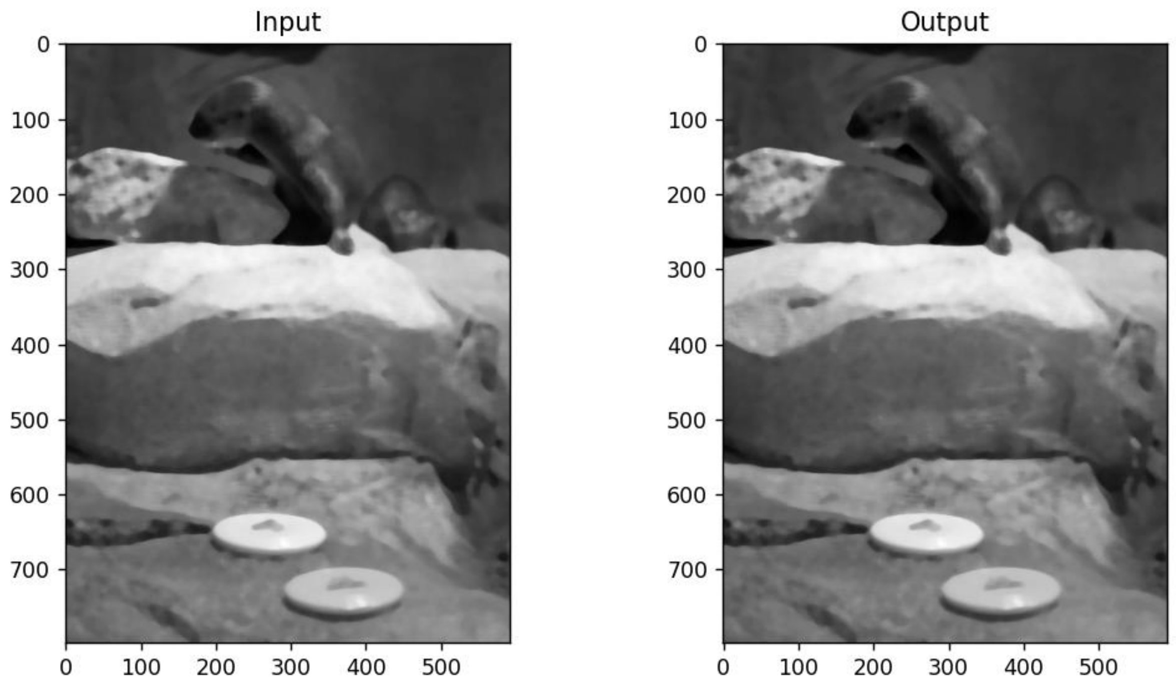


Image 4:

Operation Identified: Motion Blur

Reasoning:

- The image was processed using a Gaussian blur with a kernel size of (1, 15) and a standard deviation of 17.
- This operation replicates the effect of movement by smoothing pixel intensities along the vertical direction.
- The result is an image with elongated details and a streaking effect that mimics motion.

Effects:

- This transformation creates a visually dynamic appearance by introducing a motion-like streaking effect.
- While fine details are smoothed along the vertical axis, the overall structural context of the image is preserved.

Bonus Section:

- **Kernel/Filter Used**: Gaussian kernel with size (1, 15) and standard deviation 17.0.
- **MSE Value**: 9.466044757163179.

Result:

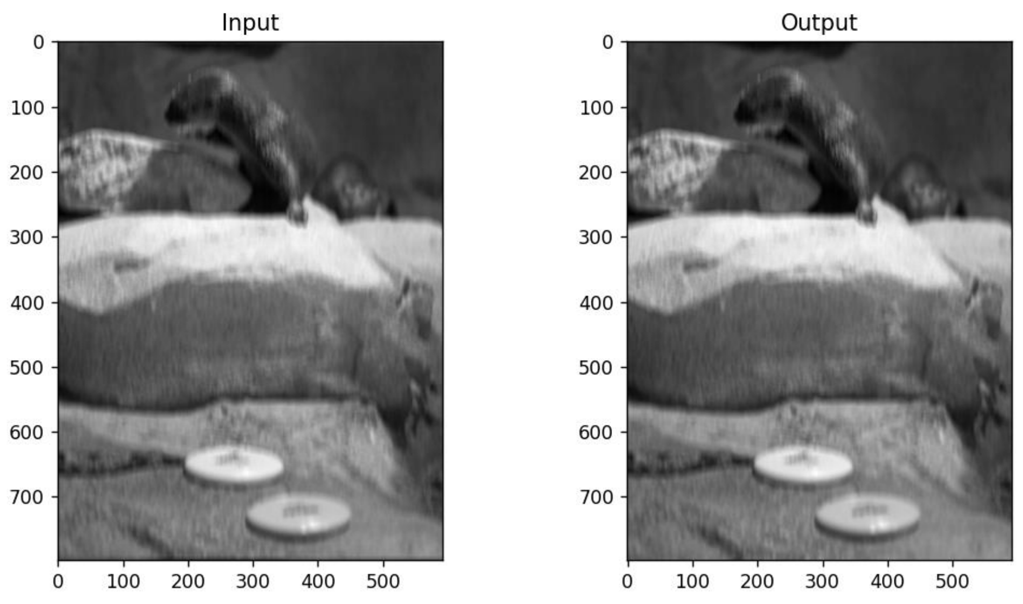


Image 5:

Operation Identified: High-Pass Filter (Using Bilateral Filtering and Brightness Offset)

Reasoning:

- A high-pass filter was applied by subtracting a bilateral-filtered version of the image from the original and adding a brightness offset of 127.
- The bilateral filter reduced noise while preserving edges, and the subtraction emphasized high-frequency details such as edges and fine textures.
- The resulting image shows enhanced sharpness and pronounced edges.

Effects:

- This operation highlights edges and fine textures, making them more distinct.
- It creates a sharper image with a clearer distinction between regions of varying intensities.

Bonus Section:

- **Bilateral Filter Settings**: Diameter = 15, SigmaColor = 240, SigmaSpace = 240.
- **Offset Used**: 127.
- **MSE Value**: 12.582006113256114.

Result:

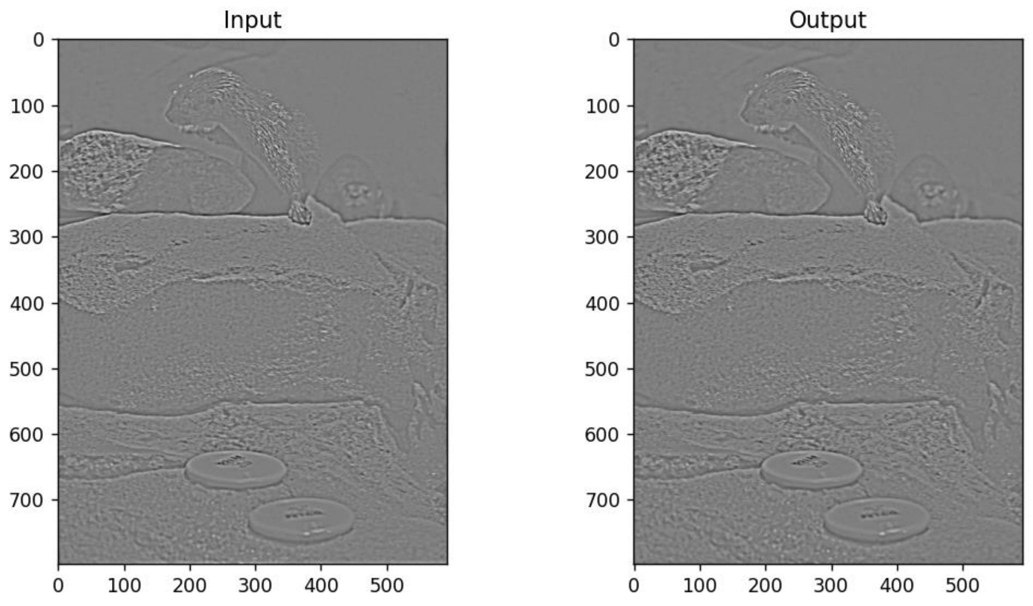


Image 6:

Operation Identified: Edge Detection (Horizontal Laplacian Kernel)

Reasoning:

- A custom Laplacian kernel was applied to detect horizontal edges by calculating gradients and highlighting transitions in intensity.
- The kernel emphasized horizontal differences in pixel values, making edges and boundaries between regions of different brightness levels more pronounced.
- The resulting image has a high-contrast appearance, effectively outlining the structures.

Effects:

- This operation highlights horizontal edges and boundaries, providing clear definitions of transitions in intensity.
- It creates a high-contrast output that enhances the visual structure of the image.

Bonus Section:

- **Kernel/Filter Used**: Custom Laplacian kernel designed to emphasize horizontal differences.
- **MSE Value**: 1.2554401205716996.

Result:

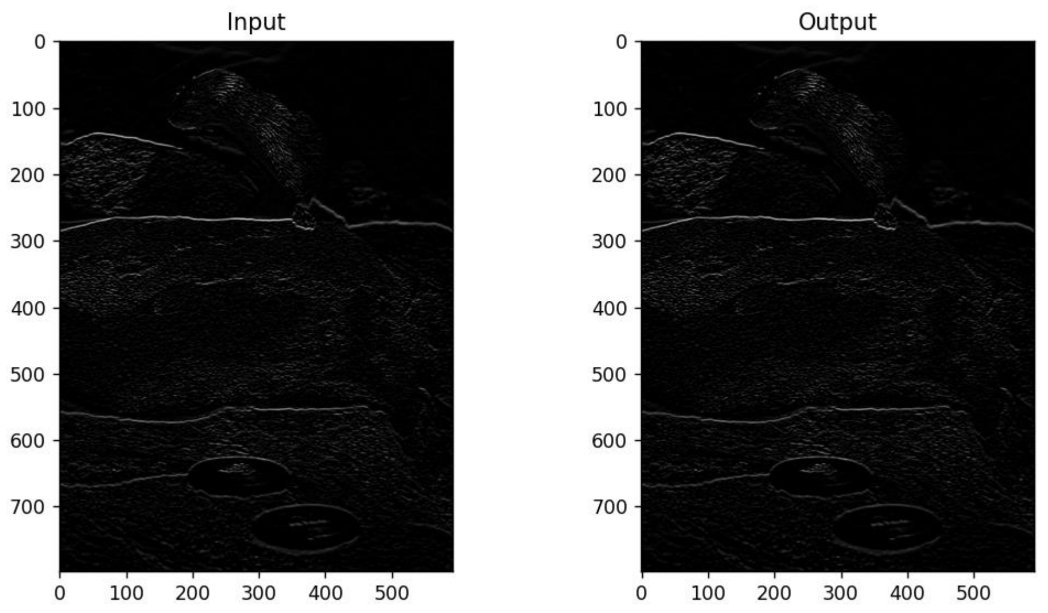


Image 7:

Operation Identified: Split-and-Swap Transformation

Reasoning:

- The image was transformed by splitting it into top and bottom halves and swapping their positions.
- This spatial manipulation rearranged the content of the image, resulting in a mirrored effect along the vertical direction.
- The pixel intensities remain unchanged, but the structure and perception of the image are significantly altered.

Effects:

- This operation creates a visually distinct rearrangement by flipping the image's vertical halves.
- It demonstrates how spatial transformations can alter the arrangement and perception of an image without modifying pixel values.

Bonus Section:

- **Operation Used**: Split and swap of the top and bottom halves of the image.
- **MSE Value**: 1.1819857921831607.

Result:

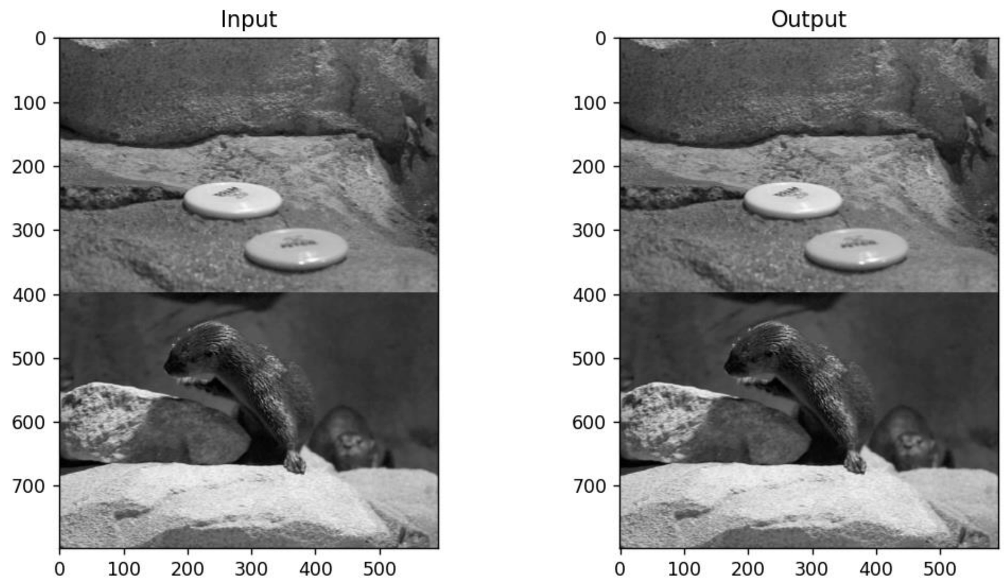


Image 8:

Operation Identified: Identity Filter (No Changes)

Reasoning:

- The image was left unaltered as a reference point for comparison.
- This identity transformation ensures that the original image is preserved in its exact form.
- It serves as a baseline to validate the accuracy and impact of the other image processing techniques.

Effects:

- The resulting image retains all original features and details without any modifications.
- It provides a consistent reference for assessing the changes introduced by other transformations.

Bonus Section:

- **Operation Used**: No transformation applied (identity filter).
- **MSE Value**: 0.3464742938427149.

Result:

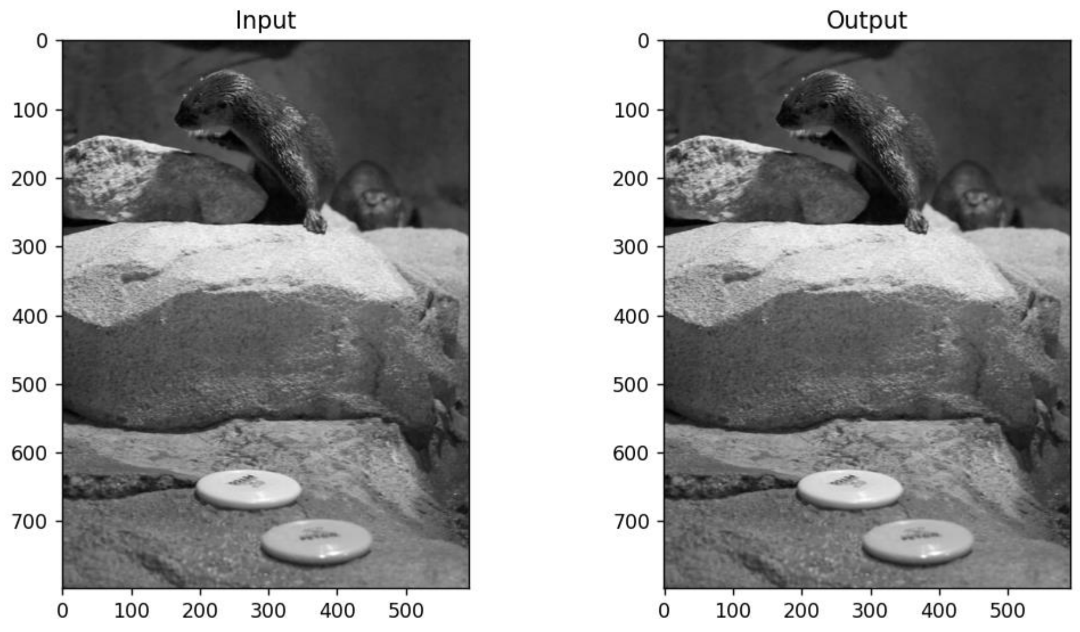


Image 9:

Operation Identified: Weighted Sharpening

Reasoning:

- A sharpening effect was achieved by combining the original image with a Gaussian blurred version using weighted addition.
- The original image was weighted at 2.1, while the blurred image was weighted at -1.1.
- This process enhances edges and details by emphasizing differences between neighboring pixel intensities.

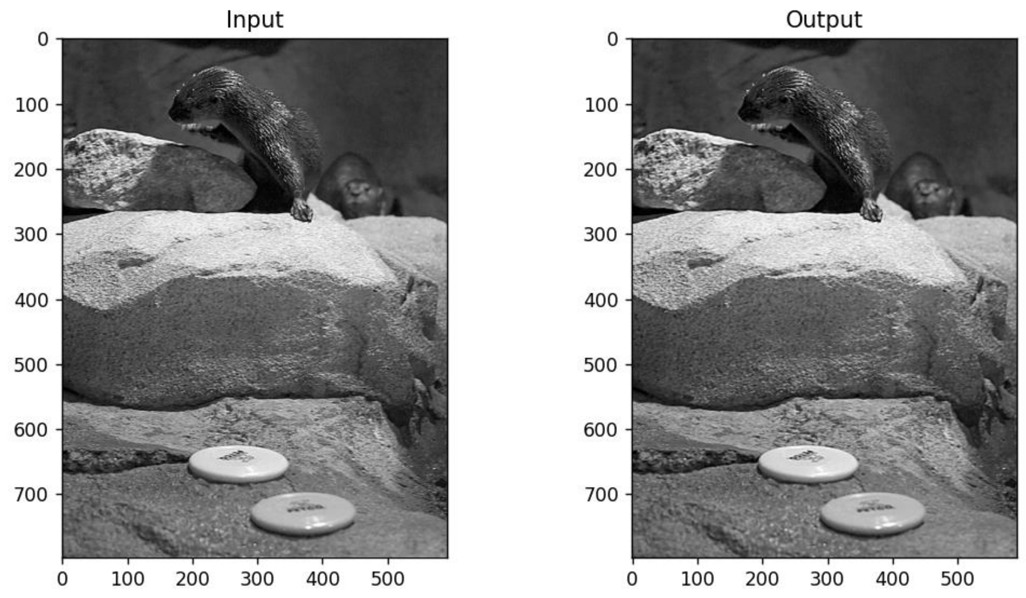
Effects:

- The resulting image appears sharper and more defined, with improved clarity and edge contrast.
- Fine textures and structures are highlighted, making the image visually more dynamic.

Bonus Section:

- **Gaussian Blur Kernel:** (9, 9).
- **Weights Used:** Original = 2.1, Blurred = -1.1.
- **MSE Value:** 20.06326839057102.

Result:



Problem 2 – Bilateral Filtering Analysis

Part 1:

Algorithm Explanation

1. **Spatial Weight Precomputation (gs)**:
   - The spatial weight depends only on the pixel's position relative to the center of the window.
   - It is computed once using the Gaussian function for a fixed window size:

```
offset = np.arange(-radius, radius + 1)
x, y = np.meshgrid(offset, offset)
gs = np.exp(-(x**2 + y**2) / (2 * stdSpatial**2))
```

2. **Pixel-by-Pixel Filtering**:
   - For each pixel in the image, define a local window of size (2*radius + 1) x (2*radius + 1). Ensure that the window doesn't exceed the image boundaries using:

```
r_min = max(i - radius, 0)
r_max = min(i + radius + 1, rows)
c_min = max(j - radius, 0)
c_max = min(j + radius + 1, cols)
```

3. **Intensity Weight Calculation (gi)**:
   - Compute intensity weights for the window dynamically based on the intensity difference between the center pixel and its neighbors:

```
gi = np.exp(-((window - im[i, j])**2) / (2 * stdIntensity**2))
```

4. **Combine Weights**:
   - Multiply the spatial (gs) and intensity (gi) weights element-wise:

```
combined_weights = gi * gs_window
```

   - Normalize the weights so their sum equals 1:

```
combined_weights /= combined_weights.sum()
```

5. **Weighted Average Calculation**:
   - Use the normalized weights to compute the new pixel value as a weighted sum of the pixel intensities in the window:

```
cleanIm[i, j] = np.sum(combined_weights * window)
```

6. **Boundary Handling**:
   - For pixels near the edges, the window size is reduced to fit within the image boundaries. This avoids out-of-bound errors while maintaining consistency in filtering.
7. **Final Output**:
   - Clip the resulting pixel values to the range [0, 255] and convert them back to uint8:

```
cleanIm = np.clip(cleanIm, 0, 255).astype(np.uint8)
```

Part 2:

1. Taj Mahal Image

Input Characteristics:

- This image posed a challenge due to significant Gaussian noise distributed across the structure and background.

Parameters Used:

- **Radius**: 7
- **stdSpatial**: 80
- **stdIntensity**: 30

Explanation of Parameter Choice:

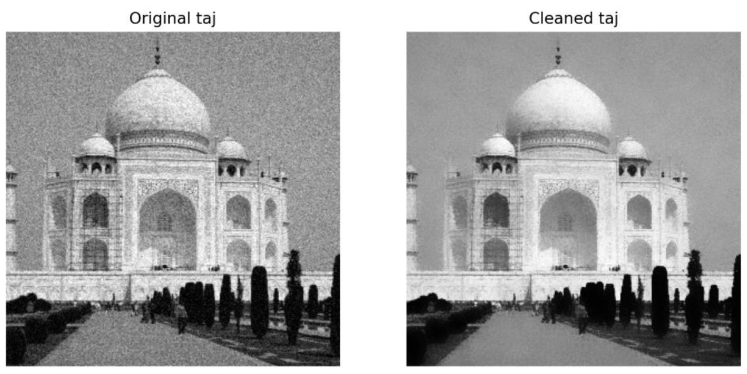
- The large spatial sigma value allowed smoothing over broader areas of the image, effectively reducing noise in the background and flat regions.
- The intensity sigma value of 30 struck a balance between denoising and preserving fine details, ensuring that edges and textures remained intact.

Observations:

- The noise was significantly reduced across the image, leading to a cleaner and more visually appealing result.
- Architectural features, such as the dome and pillars, were preserved, demonstrating the edge-preserving capability of the bilateral filter.

Result:

- The filtered image is much cleaner, with reduced noise and well-maintained structural details.



2. Noisy Gray Image

Input Characteristics:

- This image contains heavy salt-and-pepper noise distributed across two distinct regions: black and white.

Parameters Used:

- **Radius**: 7
- **stdSpatial**: 25
- **stdIntensity**: 100

Explanation of Parameter Choice:

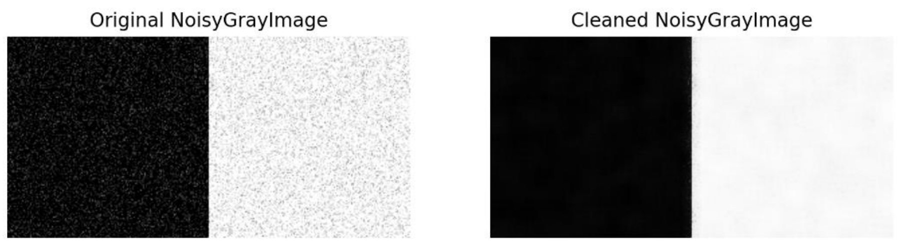
- A moderate spatial sigma ensured localized smoothing, limiting the filter's influence to nearby pixels.
- A large intensity sigma was crucial for preserving the sharp transition between the black and white regions, as it allowed greater intensity differences to be retained.

Observations:

- The noise was effectively removed, resulting in smooth black and white regions.
- The boundary between the two regions was preserved, showcasing the filter's ability to denoise while retaining critical edges.

Result:

- The filtered image is denoised with well-defined regions and minimal artifacts, achieving the desired outcome.



3. Balls Image

Input Characteristics:

- The image exhibited visible JPEG compression artifacts and minor noise.

Parameters Used:

- **Radius:** 6
- **stdSpatial:** 10
- **stdIntensity:** 9

Explanation of Parameter Choice:

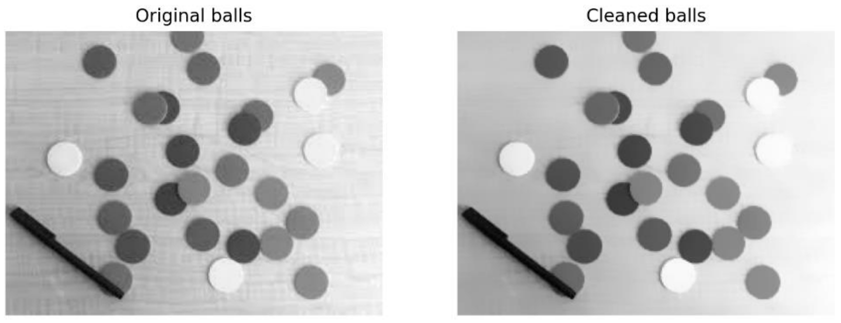
- A small spatial sigma focused the smoothing effect on local areas, avoiding distortion of finer details.
- A low intensity sigma value helped reduce blocky compression artifacts without blending the edges of the balls.

Observations:

- Compression artifacts were moderately reduced, resulting in smoother transitions across the image.
- The shapes of the balls were preserved, and the image appears cleaner overall.

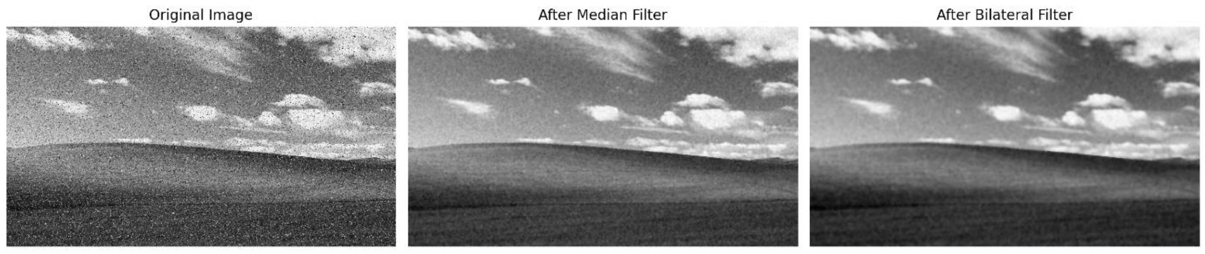
Result:

- The filtered image is visually cleaner, with reduced artifacts and well-maintained object shapes.



Problem 3 – Fix me! (40 points):

<u>Section a:</u>



1. Median Filter

- Removes salt-and-pepper noise by replacing each pixel's value with the median of the neighboring pixel values in the filter window.
- **Parameters Chosen**:
  - Kernel Size: 5
  - Explanation:
    - A moderate kernel size was selected to balance between noise removal and retaining details. We observe that larger sizes overly blur fine structures.
- **Results**:
  - Significant reduction in noise, especially in regions with high intensity variation.

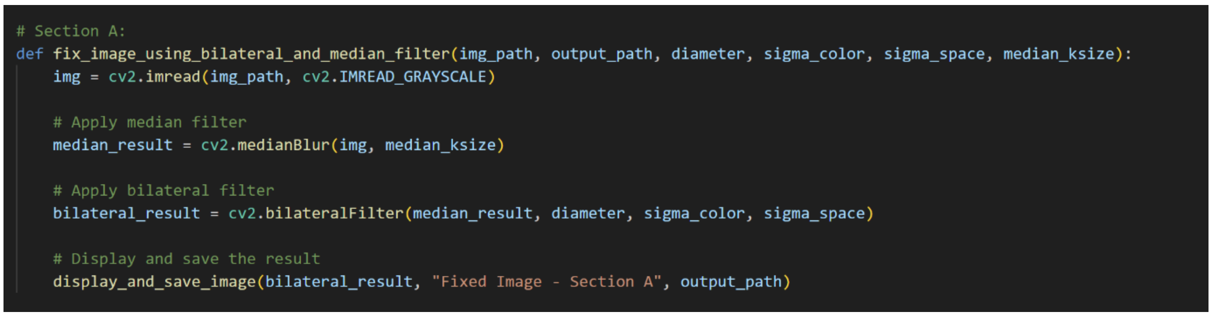
2. Bilateral Filter

- Further smoothens the image while preserving edges by weighting pixels based on spatial and intensity differences.
- **Parameters Chosen**:
  - **Diameter**: 9
  - **Sigma Color**: 80
    - A value of 80 ensures smoothening across similar intensities without over-blurring edges.
  - **Sigma Space**: 80

        - This value limits the filter's effect to a reasonable spatial distance, ensuring localized smoothing.

- **Explanation:**
    - These parameter values were chosen to balance the reduction of noise with the preservation of edges. The moderate values for sigma_color and sigma_space ensure that edges are preserved while still achieving a visible reduction in noise.
- **Results:**
    - Smoothened the sky and ground areas, effectively reducing the residual noise left after the Median Filter.
    - Preserved important structural edges, such as the horizon line and cloud outlines.

Relevant Code Snippet:

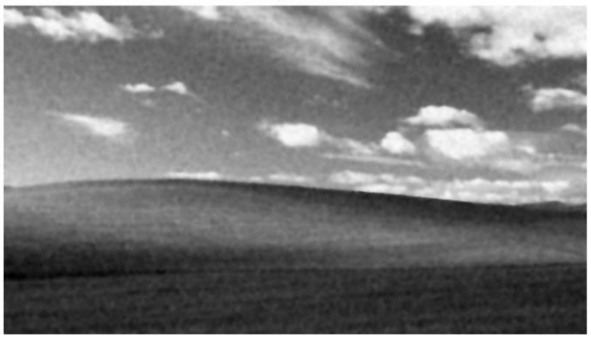


```python
# Section A:
def fix_image_using_bilateral_and_median_filter(img_path, output_path, diameter, sigma_color, sigma_space, median_ksize):
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)

    # Apply median filter
    median_result = cv2.medianBlur(img, median_ksize)

    # Apply bilateral filter
    bilateral_result = cv2.bilateralFilter(median_result, diameter, sigma_color, sigma_space)

    # Display and save the result
    display_and_save_image(bilateral_result, "Fixed Image - Section A", output_path)
```

To sum up:

- Original Image: Noisy input with visible salt-and-pepper and Gaussian-like noise.
- After Median Filter: Noise significantly reduced, but some blurring is observed in edges.
- After Bilateral Filter: Enhanced smoothness with edges preserved and further noise reduction achieved.

Fixed Image - Section A

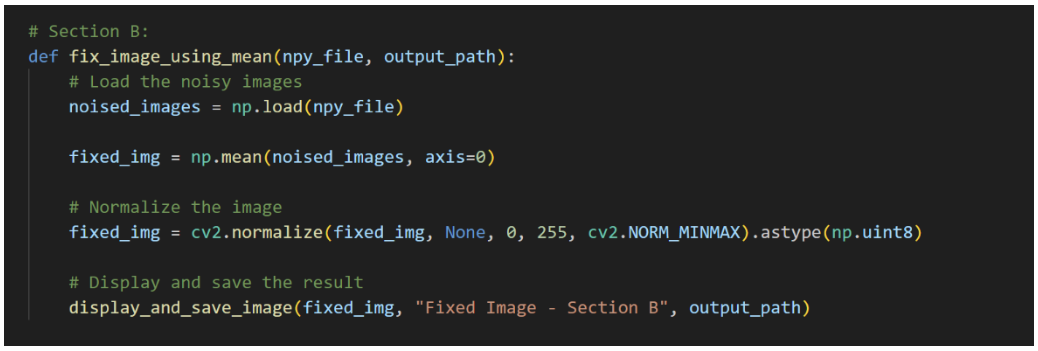


Section b:

In this section, we use the noised_images.npy to reduce the noise.

The process begins by loading the .npy file into a 3D NumPy array where each slice represents a noisy image. We then compute the mean across the first axis, which aggregates all the noisy images to form a single denoised image. To ensure the pixel values of the resulting image fall within the standard 8-bit grayscale range [0, 255], we normalize the image using OpenCV's cv2.normalize function. The cleaned image is finally displayed and saved for further evaluation.

Relevant Code Snippet:



```
# Section B:
def fix_image_using_mean(npy_file, output_path):
    # Load the noisy images
    noised_images = np.load(npy_file)

    fixed_img = np.mean(noised_images, axis=0)

    # Normalize the image
    fixed_img = cv2.normalize(fixed_img, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

    # Display and save the result
    display_and_save_image(fixed_img, "Fixed Image - Section B", output_path)
```

Cleaned Image After Noise Reduction Using Mean Averaging:

Fixed Image - Section B