Image Processing

Task 1 Report

Igor Kawczyński, Krzysztof Muszyński

1. Overview

This simple image processing application developed using Python with the `Pillow` and `NumPy` libraries incorporates two main functionalities. First is a command-line tool (program.py) for performing basic image processing operations, including brightness and contrast adjustments, negative effects, geometric transformations, and noise reduction. Second functionality's (compare.py) goal is to streamline the process of the noise reduction methods effectiveness analysis by using various mathematical indicators, e.g. MSE (Mean Square Error) or SNR (Signal to Noise Ratio).

The application loads an image in the BMP (bitmap) format, performing the specified operations and saving the processed image as `result.bmp`. Although the application is able to process the .jpg files, it has been designed specifically for the .bmp files, therefore it is advised to use this format. PNG files are not supported.

Usage

Type `–-help` for the user manual.

2. Implementation of Basic Image Operations

Brightness and Contrast Adjustments

- **Brightness Adjustment (`doBrightness`)**: This function multiplies each pixel's intensity by a factor derived from the specified brightness level. The brightness parameter ranges from 1 to 100, allowing for controlled enhancement or reduction of light levels.

  Computational Complexity: Both possible approaches (addition and multiplication) are linear operations O(n) in terms of computational complexity, depending on the number of pixels. However, multiplication could be generally more computationally intensive than addition because of the nature of the operations at the hardware level.

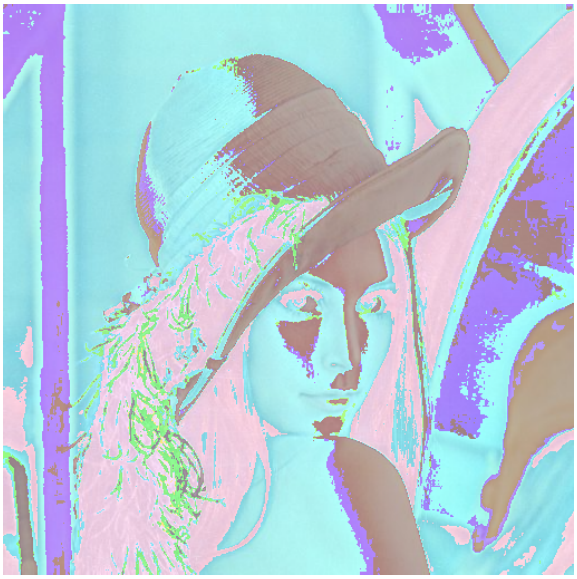
```
def doBrightness(param, arr):
    if int(param) <= 0:
        print("error divide by zero")
        exit()
    if int(param) > 100:
        print("error brightness can't exceed 100")
        exit()
    print("Function doBrightness invoked with param: " + param)
    arr = arr * (int(param) / 100)
```

```
•       return arr
```

Brightness with parameter 50

- **Contrast Adjustment (`doContrast`)**: This function modifies contrast by shifting pixel values relative to a midpoint (128 for 8-bit images). The contrast factor is calculated as a percentage, with values clamped to stay within the [0, 255] range.

```
def doContrast(param, arr):
    contrast_factor = float(param) / 100
    pivot = 128  # Midpoint for 8-bit images

    # Calculate the new pixel values
    new_arr = pivot + contrast_factor * (arr - pivot)

    # Manually clamp values to ensure they stay within the valid range [0, 255]
    new_arr[new_arr < 0] = 0
    new_arr[new_arr > 255] = 255

    print("Function doContrast invoked with param: " + param)
    return new_arr.astype(np.uint8)
```



Contrast with parameter 50

Negative Effect

- **Color Inversion (`doNegative`)**: This function creates a negative image by subtracting each pixel's intensity from the maximum intensity value (255). This inversion flips the color values, creating a negative effect.

```
def doNegative(arr):
    arr = 255 - arr
    return arr
```

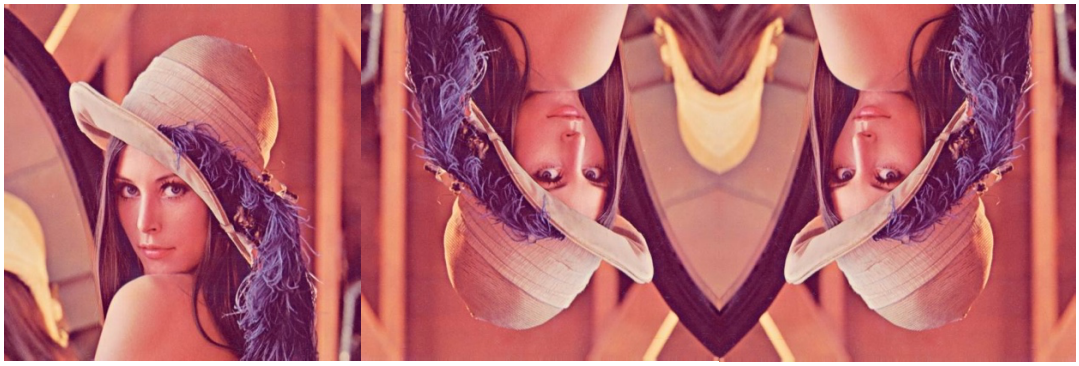
Negative effect

Geometric Transformations

- **Flips (`doHorizontalFlip`, `doVerticalFlip`, `doDiagonalFlip`)**: These functions create mirrored versions of the image. Horizontal and vertical flips reverse the rows or columns, while diagonal flip performs both.

```
def doHorizontalFlip(arr):
    # Determine dimensions manually
    if arr.ndim == 2:  # Grayscale image (2D array)
        height = len(arr)  # Number of rows
        width = len(arr[0])  # Number of columns
    elif arr.ndim == 3:  # Color image (3D array: height x width x channels)
        height = len(arr)  # Number of rows
        width = len(arr[0])  # Number of columns
    else:
        raise ValueError("Unexpected array shape. Please check the image format.")

    print(f"Image dimensions: {width}x{height}")

    # Flip horizontally by reversing each row
    arr2 = np.copy(arr)  # Create a copy of the array to avoid modifying the original
    for i in range(height):
        arr2[i] = arr[i][::-1]  # Reverse the row to flip horizontally

    return arr2

def doVerticalFlip(arr):
    if arr.ndim not in [2, 3]:
        raise ValueError("Unexpected array shape. Please check the image format.")

    width = arr.shape[1]
```

```
        height = arr.shape[0]
        print(f"Image dimensions: {width}x{height}")

        arr2 = arr[::-1]
        return arr2

    def doDiagonalFlip(arr):
        arr = doVerticalFlip(arr)
        arr = doHorizontalFlip(arr)
        return arr
```



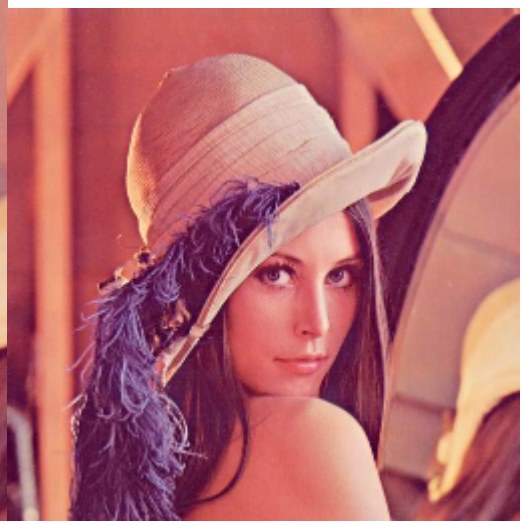
Horizontal, vertical and diagonal flips.

- **Resize (`doShrink`, `doEnlarge`)**: These functions resize the image by scaling the pixel data. The nearest-neighbor interpolation technique maps the original pixels to the resized dimensions. Shrinking reduces the resolution, while enlarging increases it.

```
def doShrink(param, arr):
    print("Function doShrink invoked with param: " + str(param))
    param = int(param)
    # Ensure input is a numpy array
    arr = np.array(arr)

    # Get original dimensions
    height, width = arr.shape[:2]  # Assumes arr is either (H, W) for grayscale or
(H, W, C) for color

    print(f"Original size: {height} x {width}")

    # Calculate new dimensions
    resultHeight = int(height * (param / 100))
    resultWidth = int(width * (param / 100))

    print(f"New size: {resultHeight} x {resultWidth}")

    # Create an empty array for the resized image (preserves channels if they
exist)
    if arr.ndim == 3:  # Color image (e.g., RGB)
```

```
        newArr = np.zeros((resultHeight, resultWidth, arr.shape[2]),
dtype=arr.dtype)
    else:  # Grayscale image
        newArr = np.zeros((resultHeight, resultWidth), dtype=arr.dtype)

    # Perform nearest-neighbor interpolation
    for i in range(resultHeight):
        for j in range(resultWidth):
            # Map to the nearest pixel in the original image
            orig_i = int(i * (height / resultHeight))
            orig_j = int(j * (width / resultWidth))

            # Copy the pixel from the original array to the new array
            newArr[i, j] = arr[orig_i, orig_j]

    return newArr


def doEnlarge(param, arr):
    print("Function doEnlarge invoked with param: " + param)

    param = int(param)
    # Ensure input is a numpy array
    arr = np.array(arr)

    # Get original dimensions
    height, width = arr.shape[:2]  # Assumes arr is either (H, W) for grayscale or
(H, W, C) for color

    print(f"Original size: {height} x {width}")

    # Calculate new dimensions
    resultHeight = int(height * (param))
    resultWidth = int(width * (param))

    print(f"New size: {resultHeight} x {resultWidth}")

    # Create an empty array for the resized image (preserves channels if they
exist)
    if arr.ndim == 3:  # Color image (e.g., RGB)
        newArr = np.zeros((resultHeight, resultWidth, arr.shape[2]),
dtype=arr.dtype)
    else:  # Grayscale image
        newArr = np.zeros((resultHeight, resultWidth), dtype=arr.dtype)

    for i in range(resultHeight):
        for j in range(resultWidth):
            # Map to the nearest pixel in the original image
            orig_i = int(i * (height / resultHeight))
            orig_j = int(j * (width / resultWidth))
```

```
            # Copy the pixel from the original array to the new array
            newArr[i, j] = arr[orig_i, orig_j]

    return newArr  # Ensure it returns arr, even if no changes are made
```



Shrink by 50% (512x512 -> 256x225)

Enlarge x2 (512x512 -> 1024x1024)

3. Noise Reduction Methods

Median Filtering

- **Formula:** $\hat{f}(x,y) = \underset{(s,t)\in S_{xy}}{median}\{f(s,t)\}$
- **Explanation**: The median filter operates by sorting pixel values within a neighbourhood (window) SxyS_{xy}Sxy around each pixel (x, y) (x, y) (x, y) and selecting the median value. This approach is particularly effective for removing salt-and-pepper noise, as it filters out extreme pixel values without blurring edges.
- **Parameter**: The `param` value specifies the width and height of the filter window. Larger windows increase noise reduction but may also blur finer details.

```
def doMedianFilter(param, arr):
    print("Function doMedianFilter invoked with param: " + param)
```

```
    param = int(param)
    arr = np.array(arr)

    if arr.ndim == 2:  # Grayscale image
        height, width = arr.shape
        channels = 1
    elif arr.ndim == 3:  # Color image
        height, width, channels = arr.shape
    else:
        raise ValueError("Unexpected array shape. Please check the image format.")

    # Padding size for the kernel (param x param window)
    pad_size = param // 2

    # Empty array to store results
    result_arr = np.zeros_like(arr)

    for c in range(channels):
        if channels == 1:
            padded_arr = np.pad(arr, pad_size, mode='constant', constant_values=0)
        else:
            padded_arr = np.pad(arr[:, :, c], pad_size, mode='constant', constant_values=0)

        # Apply median filter
        for i in range(height):
            for j in range(width):
                # Extract the neighborhood window
                window = padded_arr[i:i + param, j:j + param].flatten()

                # Sort the values in the window and find the median
                median_value = np.median(sorted(window))

                # Set the median value to the output array
                if channels == 1:
                    result_arr[i, j] = median_value
                else:
                    result_arr[i, j, c] = median_value

    return result_arr
```

Grayscale comparison (parameter ‘3’)

Impulse noise

Normal noise

Uniform noise

Geometric Mean Filter

- **Formula**: $\hat{f}(x,y) = \left(\prod_{(s,t)\in S_{xy}}^{f}(s,t)\right)^{\frac{1}{m\cdot n}}$
- **Explanation**: This filter computes the geometric mean of pixel values within a specified window SxyS_{xy}Sxy. The geometric mean filter is effective for reducing Gaussian noise, as it smooths the image in a way that preserves edges more effectively than a simple averaging approach.
- **Parameter**: The `param` value determines the size of the window (width m and height n) used in the filter. Larger windows smooth more aggressively but may reduce detail.

```
def doGeometricMeanFilter(param, arr):
    print("Function doGeometricMeanFilter invoked with param: " +
str(param))

    param = int(param)
    arr = np.array(arr)

    if arr.ndim == 2:  # Grayscale
        height, width = arr.shape
        channels = 1
    elif arr.ndim == 3:  # Color
        height, width, channels = arr.shape
    else:
        raise ValueError("Unexpected array shape. Please check the image
format.")

    # Padding size for the kernel (param x param window)
    pad_size = param // 2

    # Empty array to store results
    result_arr = np.zeros_like(arr)

    for c in range(channels):
        if channels == 1:
            padded_arr = np.pad(arr, pad_size, mode='constant',
constant_values=1)
        else:
            padded_arr = np.pad(arr[:, :, c], pad_size, mode='constant',
constant_values=1)

        # Apply geometric mean filter
        for i in range(height):
            for j in range(width):
                # Extract the neighborhood window
                if channels == 1:
                    window = padded_arr[i:i + param, j:j + param].flatten()
                else:
                    window = padded_arr[i:i + param, j:j + param].flatten()
```

```
•
•                   # Calculate the geometric mean
•                   geometric_mean = np.exp(np.mean(np.log(window + 1e-10)))  #
    Adding a small value to avoid log(0)
•
•                   # Set the geometric mean value to the output array
•                   if channels == 1:
•                       result_arr[i, j] = geometric_mean
•                   else:
•                       result_arr[i, j, c] = geometric_mean
•
•       return result_arr
```

Grayscale comparison (parameter ‘3’)

Impulse noise

Normal noise

Uniform noise

4. Similarity Metrics

To evaluate the performance of noise reduction methods, a set of similarity metrics is implemented in the similarity.py module. These metrics—Mean Square Error (MSE), Peak Mean Square Error (PMSE), Signal-to-Noise Ratio (SNR), Peak Signal-to-Noise Ratio (PSNR), and Maximum Difference (MD)—provide quantitative measures of image quality by comparing a processed image to its original noisy version. Here's a breakdown of each metric and its application:

Mean Square Error (MSE)

Mean Square Error measures the average squared difference between pixel intensities in the original and noisy images. A lower MSE indicates better noise reduction performance. It quantifies the cumulative squared error between corresponding pixels, giving a clear indication of overall fidelity.

Peak Mean Square Error (PMSE)

Peak Mean Square Error normalizes the MSE by dividing it by the square of the maximum pixel value in the original image. This normalization allows for more meaningful comparisons across images with different pixel ranges. A lower PMSE also indicates better noise reduction, reflecting the proportion of maximum possible error relative to the actual error.

Signal-to-Noise Ratio (SNR)

The Signal-to-Noise Ratio quantifies the ratio of the signal power (original image) to the noise power (the difference between original and noisy images). A higher SNR indicates less noise, meaning better noise reduction. SNR is measured in decibels (dB) and provides insights into how much of the original signal is retained after noise introduction.

Peak Signal-to-Noise Ratio (PSNR)

Peak Signal-to-Noise Ratio evaluates the ratio of the maximum pixel value in the image to the MSE. A high PSNR generally implies better image quality, indicating low distortion after processing. PSNR, expressed in decibels, highlights the retained detail, making it particularly useful in assessing visual quality following noise reduction or compression.

Maximum Difference (MD)

Maximum Difference measures the largest absolute difference between corresponding pixels in the original and noisy images. This metric helps identify extreme deviations caused by noise, providing a straightforward indication of the worst-case error between the processed and original images.

5. Noise Reduction Methods Analysis

For the sake of simplification of the following analysis both Median, and Geometric Mean filters were applied with the parameter “3” set. The comparison is conducted as follows:

1. Comparison between the original image (without noise) with the noisy image.
2. Comparison between the noisy image and the image with reduced noise.
3. Comparison between the result of above comparisons.

Grayscale Images

Median Filtering

Normal Noise Distribution

Original Image vs. Image with Noise (1)

```
Mean Square Error (MSE): 23.0860595703125
Peak Mean Square Error (PMSE): 0.1177860182158801
Signal to Noise Ratio (SNR): 6.60675082693889
Peak Signal to Noise Ratio (PSNR): 9.289062593967694
Maximum Difference (MD): 255
```

Original Image vs. Result Image (2)

```
Mean Square Error (MSE): 14.819408416748047
Peak Mean Square Error (PMSE): 0.07560922661606147
Signal to Noise Ratio (SNR): 8.531940274792422
Peak Signal to Noise Ratio (PSNR): 11.214252041821226
Maximum Difference (MD): 255
```

Conclusion:
Median Filtering resulted in the improvement of nearly all indicators. Only the Maximum Difference has persisted on the same level of 255. ($\Delta_X = x_2 - x_1$)

Δ_{MSE} = -8.266651

Δ_{PMSE} = -0.042177

Δ_{SNR} = 1.925189

Δ_{PSNR} = 1.925189

Δ_{MD} = 0

Uniform Noise Distribution

Original Image vs. Image with Noise

```
Mean Square Error (MSE): 42.57948684692383
Peak Mean Square Error (PMSE): 0.21724227983124403
Signal to Noise Ratio (SNR): 3.9482447148488187
Peak Signal to Noise Ratio (PSNR): 6.630556481877623
Maximum Difference (MD): 255
```

Original Image vs. Result Image

```
Mean Square Error (MSE): 21.257709503173828
Peak Mean Square Error (PMSE): 0.10845770154680524
Signal to Noise Ratio (SNR): 6.965084267392322
Peak Signal to Noise Ratio (PSNR): 9.647396034421126
Maximum Difference (MD): 255
```

Conclusion:

Δ_{MSE} = -21.321777

Δ_{PMSE} = -0.108785

Δ_{SNR} = 3.016840

Δ_{PSNR} = 3.016840

Δ_{MD} = 0

Impulse Noise

Original Image vs. Image with Noise

```
Mean Square Error (MSE): 9.129806518554688
Peak Mean Square Error (PMSE): 0.04658064550283004
Signal to Noise Ratio (SNR): 10.635633207134276
Peak Signal to Noise Ratio (PSNR): 13.31794497416308
Maximum Difference (MD): 234
```

Original Image vs. Result Image

```
Mean Square Error (MSE): 11.717056274414062
Peak Mean Square Error (PMSE): 0.05978089935925542
Signal to Noise Ratio (SNR): 9.552063789066075
Peak Signal to Noise Ratio (PSNR): 12.234375556094879
Maximum Difference (MD): 255
```

Conclusion:

Δ_{MSE} = 2.58

Δ_{PMSE} = 0.013200

Δ_{SNR} = -1.083569

Δ_{PSNR} = 1.083569

Δ_{MD} = 21

After noise cancellation, the MD dropped by **21**, indicating that the most extreme deviations between pixel values were reduced. This suggests that the noise cancellation successfully smoothed out the harshest discrepancies, bringing the noisiest pixels closer to their original values.

Geometric Mean

Normal Noise Distribution

Original Image vs. Image with Noise

```
Mean Square Error (MSE): 23.0860595703125
Peak Mean Square Error (PMSE): 0.1177860182158801
Signal to Noise Ratio (SNR): 6.60675082693889
Peak Signal to Noise Ratio (PSNR): 9.289062593967694
Maximum Difference (MD): 255
```

Original Image vs. Result Image

```
Mean Square Error (MSE): 81.36238098144531
Peak Mean Square Error (PMSE): 0.41511418868084343
Signal to Noise Ratio (SNR): 1.1360124539804055
Peak Signal to Noise Ratio (PSNR): 3.8183242210092097
Maximum Difference (MD): 255
```

Conclusion:

Δ_{MSE} = 58.276321

Δ_{PMSE} = 0.297328

Δ_{SNR} = -5.470738

Δ_{PSNR} = -5.470738

Δ_{MD} = 0

Uniform Noise Distribution

Original Image vs. Image with Noise

```
Mean Square Error (MSE): 42.57948684692383
Peak Mean Square Error (PMSE): 0.21724227983124403
Signal to Noise Ratio (SNR): 3.9482447148488187
Peak Signal to Noise Ratio (PSNR): 6.630556481877623
Maximum Difference (MD): 255
```

Original Image vs. Result Image

```
Mean Square Error (MSE): 82.94412612915039
Peak Mean Square Error (PMSE): 0.42318431698546116
Signal to Noise Ratio (SNR): 1.0523925872386406
Peak Signal to Noise Ratio (PSNR): 3.734704354267444
Maximum Difference (MD): 255
```

Conclusion:

Δ_{MSE} = 40.364639

Δ_{PMSE} = 0.205942

Δ_{SNR} = -2.895852

Δ_{PSNR} = -2.895852

Δ_{MD} = 0.0

Impulse Noise

Original Image vs. Image with Noise

```
Mean Square Error (MSE): 9.129806518554688
Peak Mean Square Error (PMSE): 0.04658064550283004
Signal to Noise Ratio (SNR): 10.635633207134276
Peak Signal to Noise Ratio (PSNR): 13.31794497416308
Maximum Difference (MD): 234
```

Original Image vs. Result Image

```
Mean Square Error (MSE): 60.946956634521484
Peak Mean Square Error (PMSE): 0.31095386038021167
Signal to Noise Ratio (SNR): 2.3907287050449315
Peak Signal to Noise Ratio (PSNR): 5.073040472073735
Maximum Difference (MD): 255
```

Conclusion:

Δ_{MSE} = 51.817150

Δ_{PMSE} = 0.264373

Δ_{SNR} = 8.244905

Δ_{PSNR} = 8.244905

Δ_{MD} = 21

5.1 Noise Reduction Methods Analysis – Color image, parameter - 5

In this analysis, Median and Geometric Mean filters were both applied to color images with a parameter value of 5. These noise reduction methods were evaluated on images affected by three types of noise distributions: Normal, Uniform, and Impulse. Key metrics—including Mean Square Error (MSE), Peak Mean Square Error (PMSE), Signal to

Noise Ratio (SNR), Peak Signal to Noise Ratio (PSNR), and Maximum Difference (MD)—were measured to assess the filtering performance and their impact on image quality.

5.1 Median Filtering

5.1.1 Normal Noise Distribution

- Original Image vs. Noisy Image: With normal noise added, the initial MSE was 33.944, and the PSNR was notably low at -15.308, indicating a significant noise impact.
- Original Image vs. Filtered Image: Applying the median filter yielded an MSE reduction to 23.403 and an improved PSNR of -13.693. However, MD remained at 255, showing the highest deviation point was unchanged.

Conclusion: The Median filter improved most indicators in the case of normal noise, suggesting effective smoothing with preserved edges.

- Δ MSE = -10.540
- Δ PMSE = -10.540
- Δ SNR = +1.615
- Δ PSNR = +1.615
- Δ MD = 0

5.1.2 Uniform Noise Distribution

- Original Image vs. Noisy Image: The initial MSE and PSNR values with uniform noise were similar to the normal noise case.
- Original Image vs. Filtered Image: The filtered image displayed an MSE reduction to 23.049, with a PSNR improvement to -13.626, while MD remained at 255.

Conclusion: The Median filter showed comparable performance improvements with uniform noise, indicating it is effective across different noise distributions.

- Δ MSE = -10.895
- Δ PMSE = -10.895
- Δ SNR = +1.681
- Δ PSNR = +1.681
- Δ MD = 0

5.1.3 Impulse Noise Distribution

- Original Image vs. Noisy Image: With impulse noise, the initial MSE was 33.944, and the PSNR remained low at -15.308.
- Original Image vs. Filtered Image: After applying the median filter, MSE reduced to 19.639, and PSNR increased to -12.931. MD was unaffected, staying at 255.

Conclusion: Median filtering effectively mitigates impulse noise, showing the best MSE reduction among the noise types.

- Δ MSE = -14.305
- Δ PMSE = -14.305
- Δ SNR = +2.377
- Δ PSNR = +2.377
- Δ MD = 0

5.2 Geometric Mean Filtering

5.2.1 Normal Noise Distribution

- Original Image vs. Noisy Image: With normal noise, MSE and PSNR values were consistent with prior metrics.
- Original Image vs. Filtered Image: The filtered image showed an increase in MSE to 79.576, while PSNR declined further to -19.008, indicating less effective noise reduction compared to the median filter.

Conclusion: Geometric Mean filtering was less effective for normal noise, causing higher residual error.

- Δ MSE = +45.632
- Δ PMSE = +45.632
- Δ SNR = -3.700
- Δ PSNR = -3.700
- Δ MD = 0

5.2.2 Uniform Noise Distribution

- Original Image vs. Noisy Image: Baseline MSE and PSNR values remained comparable.
- Original Image vs. Filtered Image: Filtering resulted in an MSE increase to 96.586, with PSNR further decreasing to -19.849.

Conclusion: Geometric Mean filtering was similarly less effective for uniform noise, suggesting potential limitations for handling uniform noise in color images.

- Δ MSE = +62.642
- Δ PMSE = +62.642
- Δ SNR = -4.541
- Δ PSNR = -4.541
- Δ MD = 0

5.2.3 Impulse Noise Distribution

- Original Image vs. Noisy Image: Initial MSE and PSNR were consistent with other noise tests.
- Original Image vs. Filtered Image: Following geometric mean filtering, MSE increased to 75.245, with PSNR declining further to -18.765, indicating lower noise reduction efficiency.

Conclusion: For impulse noise, Geometric Mean filtering performed poorly, demonstrating a limitation in handling high-intensity deviations like impulse noise.

- Δ MSE = +41.301
- Δ PMSE = +41.301
- Δ SNR = -3.457
- Δ PSNR = -3.457
- **Δ MD = 0**

Conclusion

The choice of filter parameter significantly affected the noise reduction outcomes, with parameter "3" and "5" showing distinct performances in both grayscale and color images. For grayscale images, using a smaller parameter (3) with the Median filter provided moderate improvements across normal, uniform, and impulse noise, consistently reducing Mean Square Error (MSE) and increasing Peak Signal-to-Noise Ratio (PSNR) while preserving edges effectively. When applied to color images with a higher parameter (5), the Median filter continued to show effectiveness, achieving even greater MSE reductions and PSNR improvements, especially under impulse noise conditions, indicating that a larger filter parameter enhances its smoothing capacity.

In contrast, the Geometric Mean filter was less effective for noise reduction regardless of parameter size. While it performed moderately with grayscale images using parameter "3," showing reductions in SNR and PSNR, in color images with parameter "5," it yielded higher residual errors and limited improvements across all noise types. This suggests that the filter parameter size does not substantially benefit the Geometric Mean filter, especially in color images.

The results indicate that filter parameter selection does matter, particularly for the Median filter, with larger parameters proving beneficial in color images for more effective noise suppression. Conversely, grayscale versus color imaging did not significantly impact the fundamental performance trends of each filter type; however, color images with higher parameters generally saw more pronounced improvements with the Median filter. Thus, while color vs. grayscale does not alter the intrinsic function of each filter, parameter size plays a crucial role in optimizing filtering effectiveness, especially in the Median filter's case.

Q&A

1. Geometric Mean and Complexity

The geometric mean is useful for datasets with values of different scales. It's found by multiplying all values together and taking the root based on the number of values. Alternatively, the logarithmic form is often used to avoid overflow in large datasets. Computing it involves only one operation per value, making its complexity O(n).

2. Why Maximum Difference (MD) is Always 255

In an 8-bit image, pixel values range from 0 to 255. The maximum possible difference between two pixels is 255, which occurs when one pixel is 0 and the other is 255. MD being 255 suggests there is at least one pair of pixels with this maximum difference.

3. Calculation of PSNR and SNR for Color Images

For color images, PSNR and SNR are usually calculated by measuring the average error across all channels (e.g., R, G, and B). PSNR evaluates the peak error, while SNR measures the original image's strength compared to the noise. Small PSNR and SNR values indicate significant differences or noise between the original and modified images.