Image And Video Processing Lab Experiments

Vishal Murugan DBS

Contents

1 Eye Detection using Haar Cascade Classifiers **4**
1.1 Brief Description about the Theory 4
1.2 Experimental Details 4
1.2.1 Implementation Details 4
1.2.2 Technical Approach 4
1.3 Results and Inferences 4

2 Image Transformation Techniques **6**
2.1 Brief Description about the Theory 6
2.2 Experimental Details 6
2.2.1 Geometric Transformations 6
2.2.2 Color Space Transformations 6
2.3 Results and Inferences 6

3 RGB to Grayscale Conversion **8**
3.1 Brief Description about the Theory 8
3.2 Experimental Details 8
3.2.1 Implementation Details 8
3.2.2 Technical Approach 8
3.3 Results and Inferences 9

4 Intensity Resolution Reduction **10**
4.1 Brief Description about the Theory 10
4.2 Experimental Details 10
4.2.1 Intensity Levels and Bit Depth 10
4.2.2 Implementation Details 10
4.3 Results and Inferences 11

5 Spatial Resolution Modification **12**
5.1 Brief Description about the Theory 12
5.2 Experimental Details 12
5.2.1 Resolution Levels and Effects 12
5.2.2 Implementation Details 12
5.3 Results and Inferences 12

6 Interpolation Techniques for Image Resizing **14**
6.1 Brief Description about the Theory 14
6.2 Experimental Details 14
6.2.1 Types of Interpolation 14
6.2.2 Implementation Comparison 14

6.3 Results and Inferences 15

7 Image Blurring for Smoothing and Noise Reduction 16
7.1 Brief Description about the Theory 16
7.2 Experimental Details 16
7.2.1 Implemented Blurring Techniques 16
7.3 Results and Inferences 16

8 Histogram Processing for Image Enhancement 18
8.1 Brief Description about the Theory 18
8.2 Experimental Details 18
8.2.1 Implemented Techniques 18
8.2.2 Mathematical Foundation 18
8.3 Implementation Details 18
8.3.1 Histogram Equalization 18
8.3.2 Histogram Matching 19
8.4 Results and Inferences 19

9 Frequency Domain Filtering with Box Filter 20
9.1 Brief Description about the Theory 20
9.2 Experimental Details 20
9.2.1 Implemented Approach 20
9.3 Technical Approach 20
9.3.1 Image Loading 20
9.3.2 Box Filter Creation 20
9.3.3 Frequency Domain Filtering 20
9.3.4 Output Processing 20
9.4 Results and Inferences 21

10 Frequency Domain Filtering with Gaussian Filter 22
10.1 Brief Description about the Theory 22
10.2 Experimental Details 22
10.2.1 Implemented Approach 22
10.3 Technical Approach 22
10.3.1 Image Loading 22
10.3.2 Gaussian Filter Creation 22
10.3.3 Frequency Domain Filtering 22
10.3.4 Output Processing 22
10.4 Results and Inferences 23

11 Frequency Domain Filtering with Random Filter 24
11.1 Brief Description about the Theory 24
11.2 Experimental Details 24
11.2.1 Implemented Approach 24
11.3 Technical Approach 24
11.3.1 Image Loading 24
11.3.2 Random Filter Creation 24
11.3.3 Frequency Domain Filtering 24
11.3.4 Output Processing 24
11.4 Results and Inferences 25

12 Finding DCT of a Sequence **26**
12.1 Brief Description about the Theory . . . 26
12.2 Experimental Details . . . 26
12.2.1 Implemented Approach . . . 26
12.3 Technical Approach . . . 26
12.3.1 DCT Computation . . . 26
12.3.2 Coefficient Modification . . . 26
12.3.3 IDCT Reconstruction . . . 26
12.4 Results and Inferences . . . 27

13 Sequence Energy Compaction **28**
13.1 Brief Description about the Theory . . . 28
13.2 Experimental Details . . . 28
13.2.1 Implemented Approach . . . 28
13.3 Technical Approach . . . 28
13.3.1 Energy Computation . . . 28
13.3.2 DCT Energy Distribution . . . 28
13.3.3 Effect of Coefficient Modification . . . 28
13.4 Results and Inferences . . . 29

14 Image Energy Compaction **30**
14.1 Brief Description about the Theory . . . 30
14.2 Experimental Details . . . 30
14.2.1 Implemented Approach . . . 30
14.3 Technical Approach . . . 30
14.3.1 2D DCT Computation . . . 30
14.3.2 Coefficient Cutoff . . . 30
14.3.3 Image Reconstruction via IDCT . . . 30
14.4 Results and Inferences . . . 31

15 Wavelet Decomposition **32**
15.1 Brief Description about the Theory . . . 32
15.2 Technical Background . . . 32
15.3 Implementation Details . . . 32
15.3.1 Decomposition Process . . . 32
15.4 Results and Inferences . . . 32

16 Wavelet-Based Image Denoising **34**
16.1 Brief Description about the Theory . . . 34
16.2 Technical Background . . . 34
16.3 Implementation Details . . . 34
16.3.1 Processing Steps . . . 34
16.4 Results and Inferences . . . 35

Experiment Number: 1

1 Eye Detection using Haar Cascade Classifiers

1.1 Brief Description about the Theory

This experiment demonstrates the implementation of face and eye detection using Haar Cascade Classifiers. The system uses OpenCV's pre-trained cascade classifiers to detect faces and eyes in images or real-time video streams. The Haar cascade algorithm is based on machine learning and is trained with positive and negative images to recognize specific features.

1.2 Experimental Details

1.2.1 Implementation Details

- Uses Haar Cascade Classifiers for face and eye detection.
- Converts input image to grayscale for processing.
- Implements both image and video processing capabilities.
- Draws circles around detected eyes for visualization.

1.2.2 Technical Approach

The implementation follows these key steps:

1. Image Pre-processing:
    - Converts the input image to grayscale for efficient processing.
    - Uses pre-trained Haar cascade classifiers for face and eye detection.
2. Face Detection:
    - Applies the face cascade classifier with scale factor 1.3 and minimum neighbors 4.
    - Creates regions of interest (ROI) for each detected face.
3. Eye Detection:
    - Processes each face ROI separately for eye detection.
    - Calculates eye center and radius for visualization.
    - Draws red circles (RGB: 0,0,255) around detected eyes.

1.3 Results and Inferences

The system successfully detected faces and eyes in both image and video streams. The implementation of Haar Cascade Classifiers proved effective in real-time detection. However, detection accuracy is influenced by lighting conditions and image resolution. Future improvements may include deep learning-based approaches for enhanced accuracy.

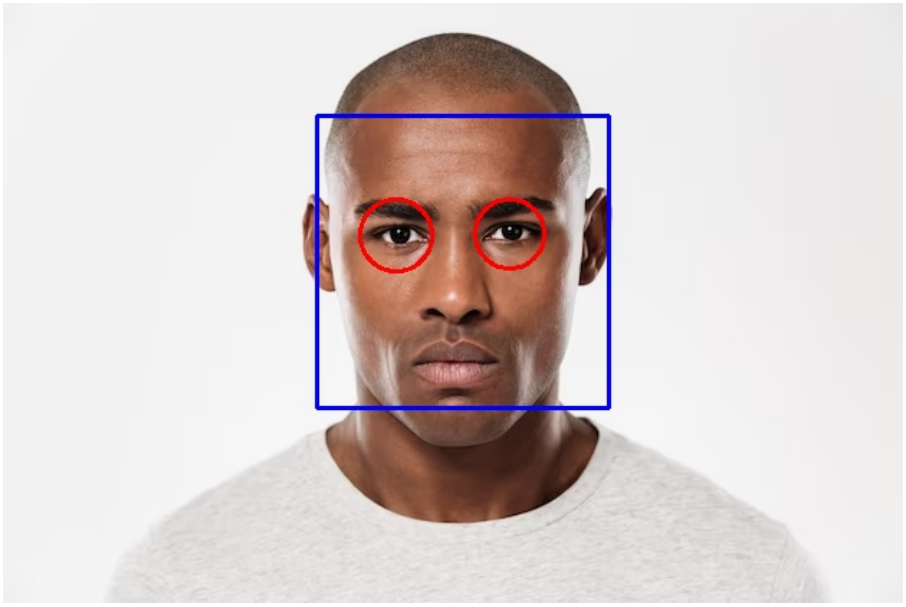


Figure 1: Eye Detection Results

Experiment Number: 2

2 Image Transformation Techniques

2.1 Brief Description about the Theory

Image transformations are fundamental operations that modify the spatial arrangement or color representation of pixels. These transformations are widely used in computer vision for preprocessing, feature extraction, and data augmentation.

2.2 Experimental Details

2.2.1 Geometric Transformations

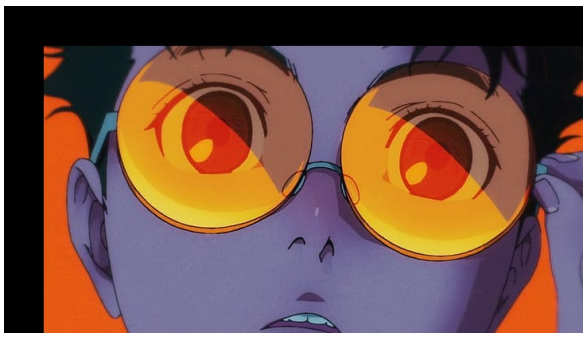
- **Translation:** Moves the image by (t_x, t_y).
- **Rotation:** Rotates the image by an angle θ.
- **Scaling:** Resizes the image by factors (s_x, s_y).
- **Affine Transformation:** Combines linear transformations with translation.

2.2.2 Color Space Transformations

- **RGB to HSV:** Separates color (Hue, Saturation) from intensity (Value).
- **RGB to LAB:** Converts the image into a perceptually uniform color space.
- **RGB to YCbCr:** Separates luminance (Y) from chrominance (Cb, Cr).

2.3 Results and Inferences

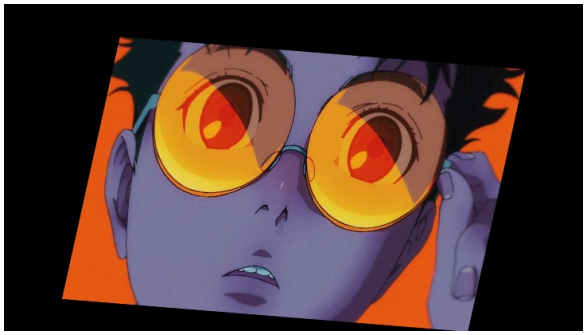
The experiment demonstrated various image transformation techniques, including geometric transformations (translation, rotation, scaling, and affine transformation) and color space conversions. These transformations enable better image processing and analysis.



(a) Translation

(b) Rotation

(c) Scaling



(d) Affine Transform

Figure 2: Image Transformation Results

Experiment Number: 3

3 RGB to Grayscale Conversion

3.1 Brief Description about the Theory

This experiment implements the luminance method for converting RGB images to grayscale using the standard formula:

$$Y = 0.299R + 0.587G + 0.114B$$

These weights are based on human perception, where the green channel has the highest contribution as the human eye is most sensitive to it.

3.2 Experimental Details

3.2.1 Implementation Details

- Uses a weighted sum of RGB channels.
- Implements the standard luminance formula.
- Preserves the perceptual luminance of the original image.

3.2.2 Technical Approach

The conversion process follows these key steps:

1. **Image Loading:**
    - Reads the input RGB image using OpenCV.
    - Converts the image to `float32` for accurate computation.
2. **Grayscale Conversion:**
    - Applies the luminance formula: $Y = 0.299R + 0.587G + 0.114B$.
    - These weights ensure an accurate grayscale representation.
    - Red channel contributes 29.9%.
    - Green channel contributes 58.7% (humans are most sensitive to green).
    - Blue channel contributes 11.4%.
3. **Output Processing:**
    - Converts the result back to `uint8` format.
    - Saves the processed grayscale image.

3.3 Results and Inferences

The grayscale conversion successfully preserves the perceptual brightness of the original image. The resulting image maintains visual details while reducing computational complexity for further processing.

(a) Input RGB Image

(b) Output Grayscale Image

Figure 3: RGB to Grayscale Conversion Results

Experiment Number: 4

4 Intensity Resolution Reduction

4.1 Brief Description about the Theory

Intensity resolution refers to the number of distinct intensity levels that an image can represent. Standard grayscale images use 8-bit representation, allowing 256 levels of intensity (0-255). Reducing the bit depth decreases the number of available intensity levels, leading to visual effects such as posterization and loss of smooth gradients.

4.2 Experimental Details

4.2.1 Intensity Levels and Bit Depth

- **8-bit:** 256 levels (0-255)
- **7-bit:** 128 levels (0-127)
- **6-bit:** 64 levels (0-63)
- **5-bit:** 32 levels (0-31)
- **4-bit:** 16 levels (0-15)
- **3-bit:** 8 levels (0-7)
- **2-bit:** 4 levels (0-3)

4.2.2 Implementation Details

The bit depth reduction process consists of the following steps:

1. **Calculate Scale Factor:** The scale factor for a given bit depth is computed as:

$$scale = \frac{255}{2^{bits} - 1}$$

2. **Quantize Pixel Values:** The pixel intensity values are adjusted based on the reduced bit depth:

$$I_{reduced} = \lfloor \frac{I_{original}}{scale} \rfloor \times scale$$

3. **Visual Effects:**
   - Reduced number of intensity levels.
   - Possible posterization effects.
   - Loss of fine intensity gradients.

4.3 Results and Inferences

The experiment successfully demonstrated the effects of reducing intensity resolution. As the bit depth decreases, the image loses fine details, leading to visible intensity banding and posterization effects. Lower bit depths significantly impact image quality, which is crucial for applications involving image compression and display optimizations.

(a) Input Grayscale Image

(b) Output Image with Reduced Intensity Resolution

Figure 4: Intensity Resolution Reduction Results

Experiment Number: 5

5 Spatial Resolution Modification

5.1 Brief Description about the Theory

Spatial resolution refers to the number of pixels used to represent an image. Higher resolution images contain more detail but require more storage, while lower resolution images lose detail but reduce file size. This experiment investigates the impact of resolution changes through downsampling and upsampling techniques.

5.2 Experimental Details

5.2.1 Resolution Levels and Effects

- **Higher resolution:** More detail, larger file size.
- **Lower resolution:** Less detail, smaller file size.
- **Common resolutions tested:** 256×256, 128×128, 64×64.

5.2.2 Implementation Details

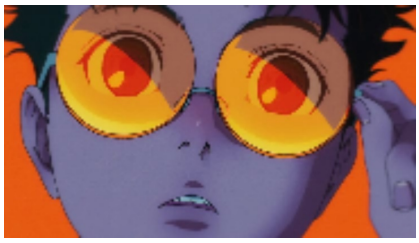
The process of modifying spatial resolution involves:

1. **Downsampling:**
    - Reduce image dimensions by integer factors.
    - Apply an anti-aliasing filter before resizing to minimize artifacts.
    - Preserve aspect ratio to avoid distortion.
2. **Quality Assessment:**
    - Visual comparison of detail retention across different resolutions.
    - Analysis of information loss due to resolution reduction.
    - Comparison of storage size for different resolutions.

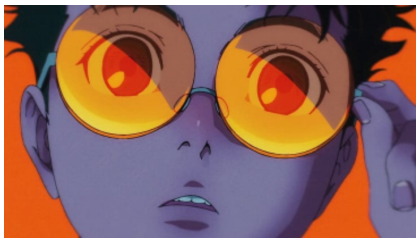
5.3 Results and Inferences

The experiment demonstrated that reducing spatial resolution leads to loss of detail and increased pixelation. Lower-resolution images retain overall structure but exhibit a noticeable decrease in fine details. The trade-off between resolution and storage size is crucial in applications such as image compression and transmission.

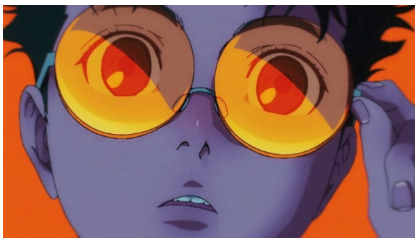
(a) Input Image



(b) Output Image 64×64



(c) Output Image 128×128



(d) Output Image 256×256

Figure 5: Spatial Resolution Variation Results

Experiment Number: 6

6 Interpolation Techniques for Image Resizing

6.1 Brief Description about the Theory

Interpolation is a fundamental technique in image processing used for resizing and reconstructing images. It estimates pixel values based on neighboring pixels, ensuring smoother transformations when scaling images. Different interpolation methods offer varying trade-offs between speed and quality.

6.2 Experimental Details

6.2.1 Types of Interpolation

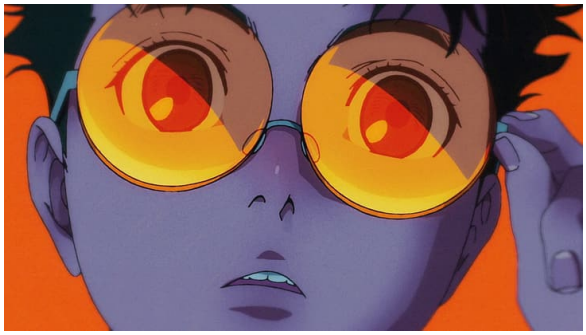
- **Nearest Neighbor:**
  - Simplest method that assigns the value of the nearest pixel.
  - Formula: $f(x, y) = f(\text{round}(x), \text{round}(y))$.
  - Fastest but produces blocky and jagged edges.
- **Bilinear Interpolation:**
  - Performs linear interpolation in both horizontal and vertical directions.
  - Uses a 2×2 neighborhood.
  - Provides smoother results compared to nearest neighbor.
  - Formula involves weighted averaging of four nearest pixels.
- **Bicubic Interpolation:**
  - Uses cubic interpolation in both directions.
  - Considers a 4×4 neighborhood, utilizing 16 surrounding pixels.
  - Produces the highest quality results, preserving fine details and reducing artifacts.
  - More computationally expensive than bilinear interpolation.

6.2.2 Implementation Comparison

- **Quality vs. Computation Time:**
  - Nearest Neighbor: Fastest but lowest quality.
  - Bilinear: Balanced trade-off between speed and quality.
  - Bicubic: Highest quality but computationally expensive.
- **Use Cases:**
  - Nearest Neighbor: Pixel art, binary images, and discrete labeling.
  - Bilinear: Real-time applications where moderate quality is acceptable.
  - Bicubic: High-quality resizing for photography, printing, and medical imaging.

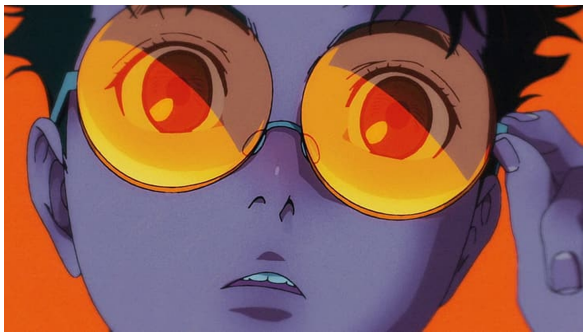
6.3 Results and Inferences

The experiment demonstrated that nearest neighbor interpolation is fast but results in visible pixelation, while bilinear interpolation improves smoothness but may blur fine details. Bicubic interpolation provides the highest visual quality, making it ideal for detailed images but requires more computation.

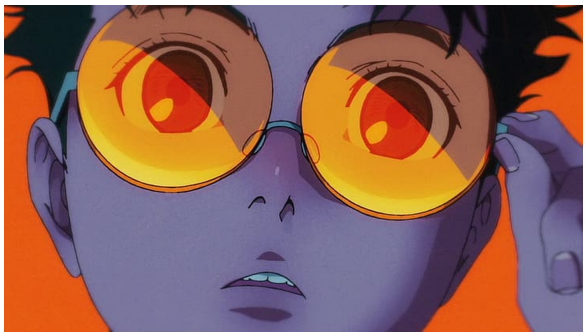


(a) Nearest Neighbor (2x)

(b) Bilinear (2x)



(c) Bicubic (2x)



(d) Lanczos (2x)

Figure 6: Interpolation Results

Experiment Number: 7

7 Image Blurring for Smoothing and Noise Reduction

7.1 Brief Description about the Theory

Image blurring is a fundamental technique used in image processing to reduce noise, smooth textures, and simulate realistic depth effects. Different blurring methods apply varying kernel operations to achieve distinct smoothing effects.

7.2 Experimental Details

7.2.1 Implemented Blurring Techniques

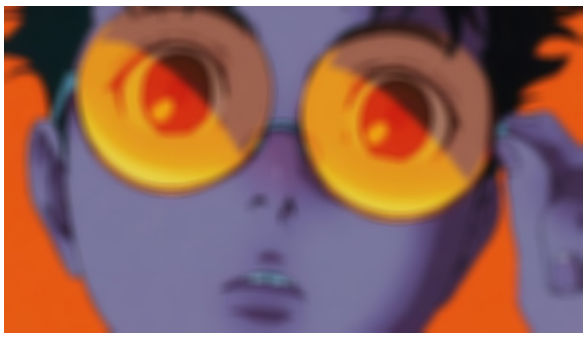
- **Gaussian Blur:**
  - Uses a Gaussian kernel for weighted averaging.
  - Kernel size: 15×15 pixels.
  - Implements the 2D Gaussian function:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

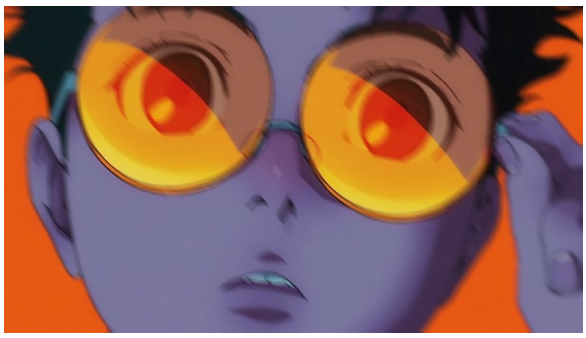
  - Effective for noise reduction while preserving edges.
- **Box Filter (Average Blur):**
  - Computes the average of all pixels in the kernel window.
  - Each pixel contributes equally to the result.
  - Fastest method but less effective for high-frequency noise.
- **Motion Blur:**
  - Simulates the effect of motion in an image.
  - Uses a directional kernel to create streaks in a specific direction.
  - The intensity and angle of blur can be controlled.
- **Focus Blur:**
  - Simulates a depth-of-field effect similar to real camera focus.
  - Applies varying blur intensities based on a depth map.
  - Useful for creating realistic depth effects in images.

7.3 Results and Inferences

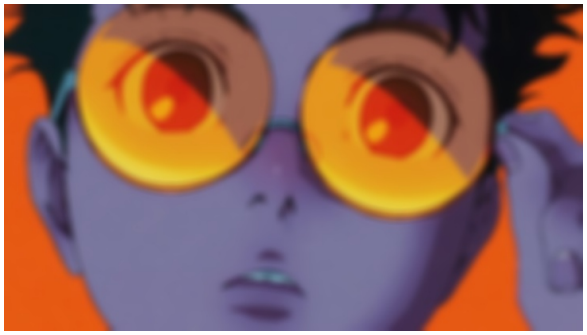
The experiment demonstrated that Gaussian Blur is highly effective for noise reduction while maintaining edge structures. The Box Filter produces uniform blurring but lacks adaptability for noise suppression. Motion Blur effectively simulates real-world motion effects, while Focus Blur mimics realistic depth perception.



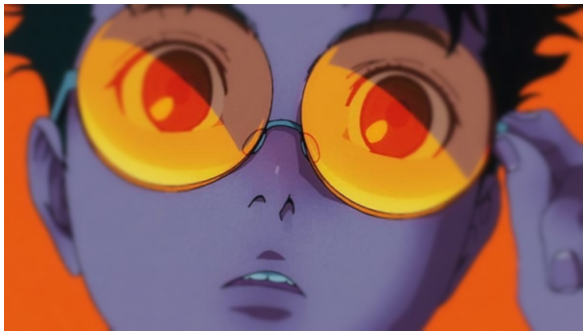
(a) Box Filter Result



(b) Motion Blur Result



(c) Gaussian Blur Input



(d) Focus Blur Output

Figure 7: Image Blurring Results

Experiment Number: 8

8 Histogram Processing for Image Enhancement

8.1 Brief Description about the Theory

Histogram processing is a crucial technique in image enhancement and analysis. It helps improve contrast, brightness, and overall visibility of details in an image. The histogram represents the frequency distribution of pixel intensities, allowing for transformations like histogram equalization and histogram matching.

8.2 Experimental Details

8.2.1 Implemented Techniques

- **Histogram Equalization:**
  - Redistributes intensity values to enhance contrast.
  - Uses the cumulative distribution function (CDF) for transformation.
  - Particularly useful for images with poor contrast.
- **Histogram Matching:**
  - Adjusts the histogram of one image to match that of another.
  - Ensures similar brightness and contrast levels between images.
  - Uses a lookup table (LUT) for pixel intensity mapping.

8.2.2 Mathematical Foundation

- **Cumulative Distribution Function (CDF):**

$$CDF(i) = \sum_{j=0}^{i} \frac{histogram(j)}{total\_pixels}$$

- **Lookup Table Generation for Matching:**

$$LUT[i] = \arg\min_{j} |CDF_{target}(j) - CDF_{source}(i)|$$

8.3 Implementation Details

8.3.1 Histogram Equalization

1. Compute the histogram of the input image.
2. Calculate the cumulative distribution function (CDF).
3. Normalize and use CDF to remap intensity values.
4. Generate the enhanced output image.

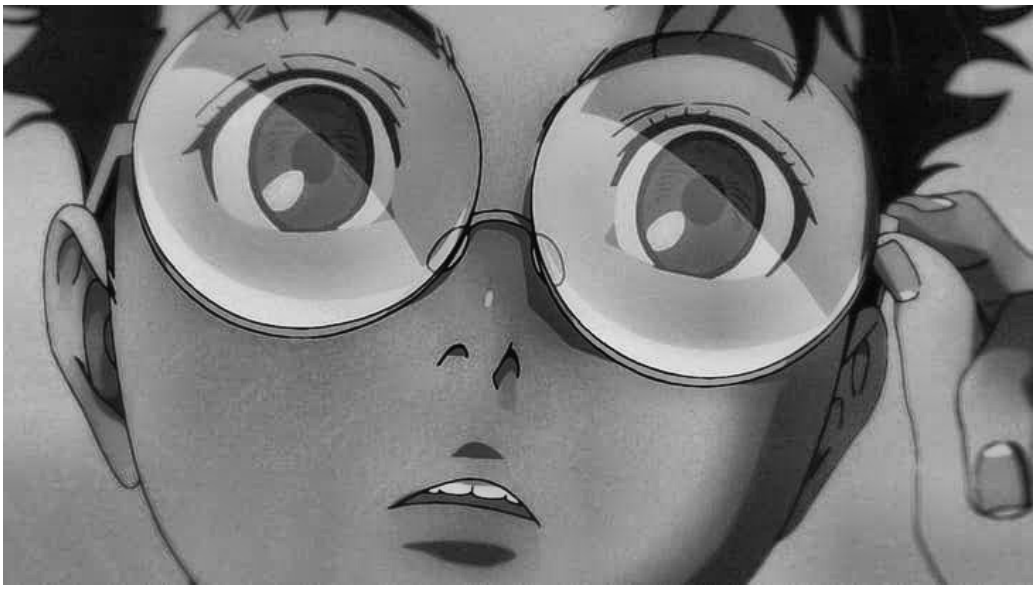
8.3.2 Histogram Matching

1. Convert both source and target images to grayscale.
2. Compute histograms and cumulative distribution functions (CDFs).
3. Generate a mapping function using a lookup table (LUT).
4. Apply the mapping to the source image to match the target histogram.

8.4 Results and Inferences

Histogram equalization successfully improves contrast by spreading intensity values across the full dynamic range. Histogram matching allows controlled modification of image brightness and contrast, making it useful for image normalization tasks.

(a) Histogram Equalization Result



(b) Adaptive Histogram Equalization Result

Figure 8: Histogram Processing Results

Experiment Number: 9

9 Frequency Domain Filtering with Box Filter

9.1 Brief Description about the Theory

Frequency domain filtering is a powerful technique in image processing where transformations are performed in the frequency domain rather than the spatial domain. The Box filter, a simple low-pass filter, is applied in the frequency domain to remove high-frequency components, effectively smoothing the image.

9.2 Experimental Details

9.2.1 Implemented Approach

- A Box filter is used to attenuate high-frequency components.
- Both the image and the filter are transformed to the frequency domain using Fast Fourier Transform (FFT).
- Filtering is performed by element-wise multiplication in the frequency domain.
- The processed image is then transformed back to the spatial domain using the Inverse FFT.

9.3 Technical Approach

The filtering process follows these key steps:

9.3.1 Image Loading

- The input image is read and converted to grayscale if necessary.
- The image is converted to `float32` for precise FFT computation.

9.3.2 Box Filter Creation

- A Box filter of size 50×50 is created in the spatial domain.
- The filter is padded to match the dimensions of the input image.

9.3.3 Frequency Domain Filtering

- The image and the Box filter are both transformed to the frequency domain using FFT.
- The frequency domain representations of the filter and the image are multiplied element-wise.
- The inverse FFT is applied to convert the filtered image back to the spatial domain.

9.3.4 Output Processing

- The resulting image is displayed and saved for analysis.

9.4 Results and Inferences

Applying the Box filter in the frequency domain effectively removes high-frequency noise and results in a smoother image. The method demonstrates how frequency domain filtering can provide efficient noise reduction and low-pass filtering.

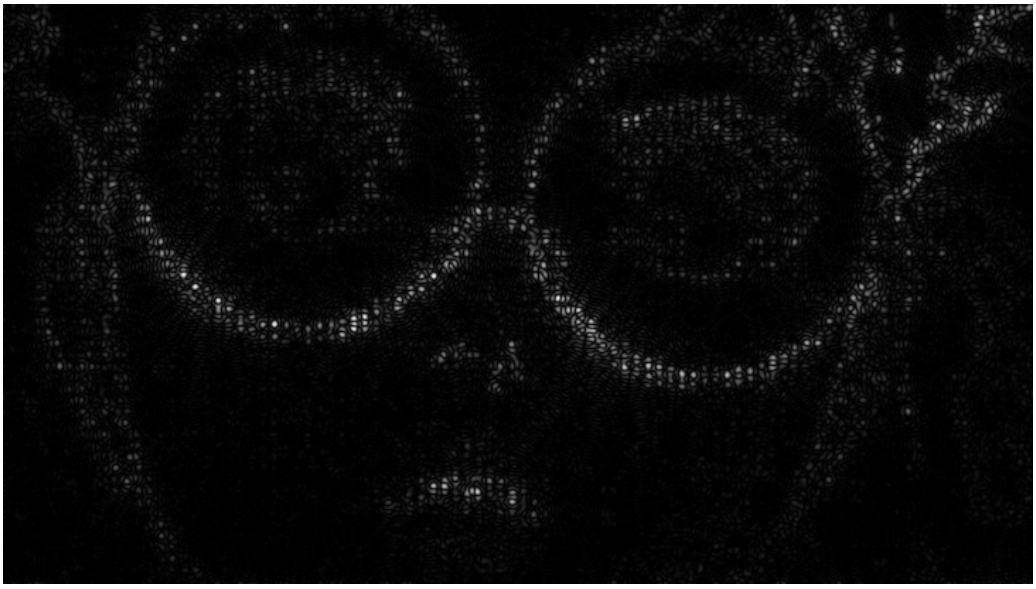


Figure 9: Frequency Domain Filtering with Box Filter

Experiment Number: 10

10 Frequency Domain Filtering with Gaussian Filter

10.1 Brief Description about the Theory

Frequency domain filtering is an essential technique in image processing that modifies image frequencies to enhance or suppress specific details. The Gaussian filter is a low-pass filter that smoothly attenuates high-frequency components, preserving essential image structures while reducing noise.

10.2 Experimental Details

10.2.1 Implemented Approach

- A Gaussian filter is designed to suppress high-frequency noise while retaining low-frequency details.
- The image and filter are transformed into the frequency domain using the Fast Fourier Transform (FFT).
- Filtering is performed through element-wise multiplication in the frequency domain.
- The processed image is reconstructed via the Inverse FFT.

10.3 Technical Approach

The filtering process follows these key steps:

10.3.1 Image Loading

- The input image is read and converted to grayscale if necessary.
- The image is converted to `float32` format for precise FFT computation.

10.3.2 Gaussian Filter Creation

- A Gaussian filter is created in the spatial domain using the standard Gaussian function:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- The filter is padded to match the dimensions of the input image.

10.3.3 Frequency Domain Filtering

- The image and the Gaussian filter are transformed into the frequency domain using FFT.
- The frequency representations of the filter and the image are multiplied element-wise.
- The inverse FFT is applied to reconstruct the filtered image in the spatial domain.

10.3.4 Output Processing

- The final processed image is displayed and saved for further analysis.

10.4 Results and Inferences

Applying the Gaussian filter in the frequency domain effectively reduces noise while maintaining smooth transitions. This method is widely used in applications like denoising, image enhancement, and edge-preserving smoothing.

Figure 10: Frequency Domain Filtering with Gaussian Filter

Experiment Number: 11

11 Frequency Domain Filtering with Random Filter

11.1 Brief Description about the Theory

Frequency domain filtering is a technique that processes an image by modifying its frequency components. Unlike standard filters (e.g., Gaussian or Box filters), a random filter applies an arbitrary transformation, which can introduce unpredictable alterations to the image structure.

11.2 Experimental Details

11.2.1 Implemented Approach

- A random filter is generated using a uniform distribution of random values.
- The random filter is created in the spatial domain and then transformed into the frequency domain using FFT.
- Image filtering is performed by multiplying the frequency representation of the image with that of the random filter.

11.3 Technical Approach

The filtering process consists of the following steps:

11.3.1 Image Loading

- The input image is read and converted to grayscale if required.
- The image is converted to `float32` format to maintain numerical precision during FFT computations.

11.3.2 Random Filter Creation

- A random filter is generated with uniform random values across the image dimensions.
- Unlike traditional filters, this filter lacks predefined characteristics and introduces unpredictable transformations.

11.3.3 Frequency Domain Filtering

- The random filter and the input image are transformed into the frequency domain using FFT.
- Element-wise multiplication is performed between the frequency representations of the image and the random filter.
- The result is transformed back to the spatial domain using the inverse FFT (IFFT).

11.3.4 Output Processing

- The final filtered image is displayed and saved for analysis.

11.4 Results and Inferences

Applying a random filter in the frequency domain results in unpredictable modifications to the image. The effects can vary from subtle changes in texture to complete distortion, making this method more suitable for artistic transformations or experimental processing rather than structured image enhancement.

Figure 11: Frequency Domain Filtering with Random Filter

Experiment Number: 12

12 Finding DCT of a Sequence

12.1 Brief Description about the Theory

The Discrete Cosine Transform (DCT) is a widely used mathematical transformation in signal processing, particularly in image compression techniques like JPEG. It transforms a sequence from the time domain into the frequency domain, helping in energy compaction and reducing redundancy.

12.2 Experimental Details

12.2.1 Implemented Approach

- Compute the DCT of a given sequence using SciPy's `fft.dct` function.
- Modify the DCT coefficients by setting the last element to zero.
- Apply the Inverse DCT (IDCT) using `fft.idct` to reconstruct the modified sequence.
- Compare the original and reconstructed sequences.

12.3 Technical Approach

The experiment follows these key steps:

12.3.1 DCT Computation

- The input sequence is transformed into the frequency domain using the Discrete Cosine Transform.
- The DCT coefficients represent frequency components, with lower indices capturing low-frequency content and higher indices capturing finer details.

12.3.2 Coefficient Modification

- The last DCT coefficient is set to zero.
- This step demonstrates how removing high-frequency components affects the reconstruction.

12.3.3 IDCT Reconstruction

- The modified DCT coefficients are transformed back into the time domain using the Inverse Discrete Cosine Transform (IDCT).
- The resulting sequence is compared with the original sequence to observe the impact of removing the last coefficient.

12.4 Results and Inferences

Removing the last DCT coefficient slightly alters the reconstructed sequence but generally retains the overall structure. This experiment demonstrates the effect of modifying frequency components and their impact on data representation.

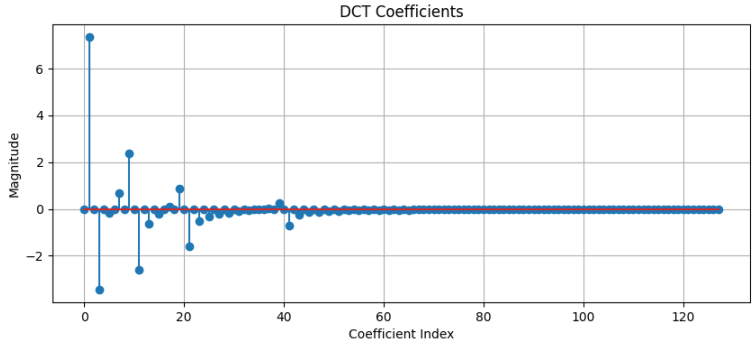


Figure 12: DCT Experiment Results

Experiment Number: 13

13 Sequence Energy Compaction

13.1 Brief Description about the Theory

The Discrete Cosine Transform (DCT) is widely used for energy compaction, concentrating most of a signal's energy into a few coefficients. This property is essential in compression techniques such as JPEG, where low-energy components can be discarded with minimal loss in quality. This experiment evaluates how energy is distributed in the original sequence and its DCT representation, including the effect of setting the last DCT coefficient to zero.

13.2 Experimental Details

13.2.1 Implemented Approach

- Compute the total energy of the original sequence.
- Perform the Discrete Cosine Transform (DCT) on the sequence.
- Calculate the energy of the DCT coefficients.
- Modify the DCT representation by setting the last coefficient to zero.
- Compute the energy of the modified DCT sequence and compare it with the original.

13.3 Technical Approach

The experiment follows these key steps:

13.3.1 Energy Computation

- The total energy of a sequence is computed as:

$$E_{\text{sequence}} = \sum_{i=0}^{N-1} x[i]^2$$

  where $x[i]$ represents the sequence values.

13.3.2 DCT Energy Distribution

- The energy of the DCT-transformed sequence is calculated similarly.
- This transformation helps visualize how energy is concentrated in specific coefficients.

13.3.3 Effect of Coefficient Modification

- The last DCT coefficient is set to zero.
- The resulting energy is recalculated to observe any loss.

13.4 Results and Inferences

This experiment demonstrates that the DCT effectively compacts energy into fewer coefficients, and modifying a single coefficient has a minimal but noticeable effect on total energy.

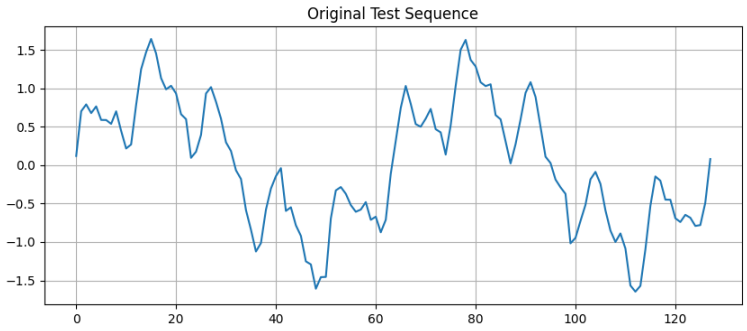


Figure 13: Sequence Energy Experiment Results

Experiment Number: 14

14 Image Energy Compaction

14.1 Brief Description about the Theory

Energy compaction is a fundamental principle in image processing and compression. The Discrete Cosine Transform (DCT) helps concentrate image energy into fewer coefficients, enabling efficient compression. By retaining only significant coefficients, we can reconstruct an image with minimal perceptual loss.

14.2 Experimental Details

14.2.1 Implemented Approach

- Compute the 2D Discrete Cosine Transform (DCT) of an image.
- Retain only 50% of the DCT coefficients (low-frequency components).
- Apply the Inverse DCT (IDCT) to reconstruct the image.
- Compare the original, DCT, and reconstructed images.

14.3 Technical Approach

The experiment follows these key steps:

14.3.1 2D DCT Computation

- The input image is transformed into the frequency domain using the 2D DCT.
- This transformation expresses the image as a sum of frequency components.
- The DCT coefficients are analyzed for energy distribution.

14.3.2 Coefficient Cutoff

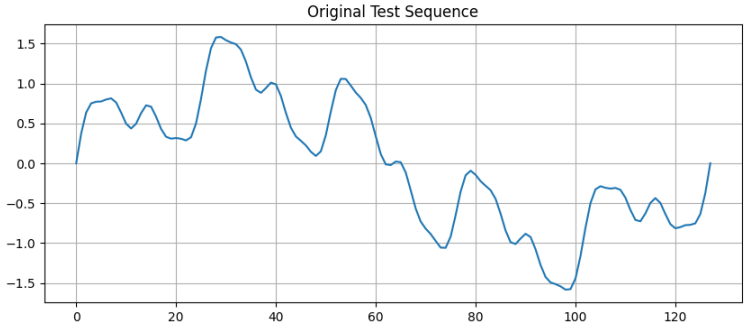
- Only the lower 50% of frequencies (high-energy components) are retained.
- The remaining coefficients are set to zero to simulate compression.

14.3.3 Image Reconstruction via IDCT

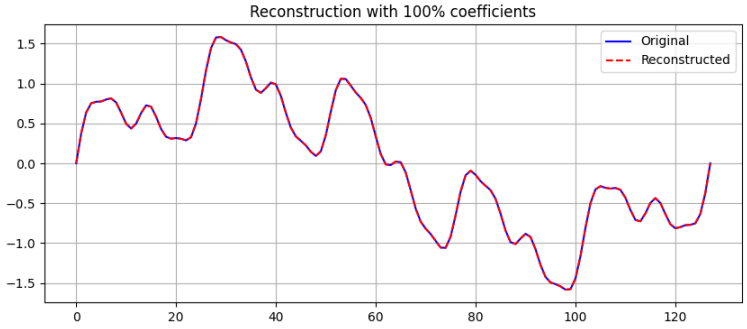
- The truncated DCT coefficients are used to reconstruct the image via IDCT.
- The reconstructed image is compared against the original.

14.4 Results and Inferences

This experiment demonstrates that a significant portion of an image's energy is concentrated in lower frequencies. Even after removing 50% of the coefficients, the reconstructed image retains most of the visual information, illustrating the effectiveness of energy compaction in compression.



(a) Original Sequence



(b) Reconstructed Sequence

Figure 14: DCT Analysis Results

Experiment Number: 15

15 Wavelet Decomposition

15.1 Brief Description about the Theory

Wavelet decomposition provides a multi-resolution analysis of images, making it a powerful tool in image compression and feature extraction. Unlike the Fourier Transform, which analyzes signals in the frequency domain, wavelets analyze images in both spatial and frequency domains, allowing localized feature representation.

15.2 Technical Background

Wavelet transforms break down an image into multiple frequency components at different scales:

- **Haar Wavelet Decomposition**: Splits an image into four sub-bands using wavelet filters.
- **Frequency Components**: Produces low-frequency (**LL**) and high-frequency components (**LH**, **HL**, **HH**).
- **Multi-Resolution Analysis**: Helps analyze the image at different levels of detail.

15.3 Implementation Details

The experiment utilizes the PyWavelets library (`pywt`) for 2D wavelet decomposition:

15.3.1 Decomposition Process

1. **Pre-processing:**
   - Convert the input image to grayscale.
   - Normalize pixel values for better numerical stability.
2. **Wavelet Transformation:**
   - Apply Haar wavelet decomposition using `pywt.dwt2()`.
   - Extract four sub-bands: LL (approximation), LH (horizontal details), HL (vertical details), HH (diagonal details).
3. **Visualization and Analysis:**
   - Display and analyze the four sub-band images.
   - Compare the original image with its wavelet decomposition.

15.4 Results and Inferences

This experiment highlights the effectiveness of Haar wavelet decomposition in representing an image's frequency components. The LL sub-band retains most of the structural information, while the LH, HL, and HH components emphasize finer details.

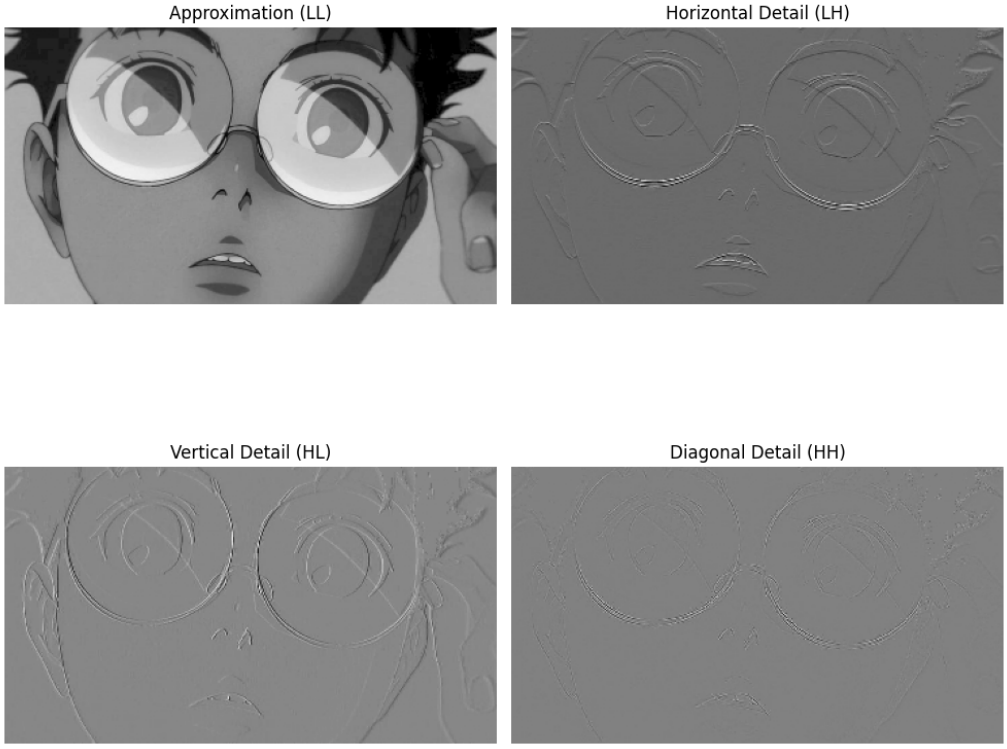


Figure 15: Wavelet Experiment Results - Decomposed Sub-bands

Experiment Number: 16

16 Wavelet-Based Image Denoising

16.1 Brief Description about the Theory

Wavelet-based denoising is a powerful technique for image noise reduction, leveraging the ability of wavelet transforms to separate low and high-frequency components. This experiment applies Haar wavelet decomposition to analyze and suppress noise while preserving image details.

16.2 Technical Background

Wavelet transforms are useful for image denoising because they allow selective modification of frequency components:

- **Wavelet Decomposition:** The image is decomposed into four sub-bands:
  - **LL (Approximation):** Low-frequency components (main structure of the image).
  - **LH (Horizontal details)**, **HL (Vertical details)**, and **HH (Diagonal details)**: High-frequency components (edges and noise).
- **Denoising Methods:**
  - **Method 1:** All detail coefficients (**LH, HL, HH**) are set to zero.
  - **Method 2:** Thresholding is applied to the detail coefficients, setting values below a defined threshold to zero.
- **Wavelet Reconstruction:** The denoised image is reconstructed using the Inverse Discrete Wavelet Transform (IDWT).
- **Quality Evaluation:** Peak Signal-to-Noise Ratio (PSNR) is computed to assess denoising effectiveness.

16.3 Implementation Details

The experiment is implemented using:

- **OpenCV:** Image loading and preprocessing.
- **PyWavelets (pywt):** Haar wavelet decomposition and reconstruction.
- **Scikit-Image:** PSNR computation for image quality evaluation.

16.3.1 Processing Steps

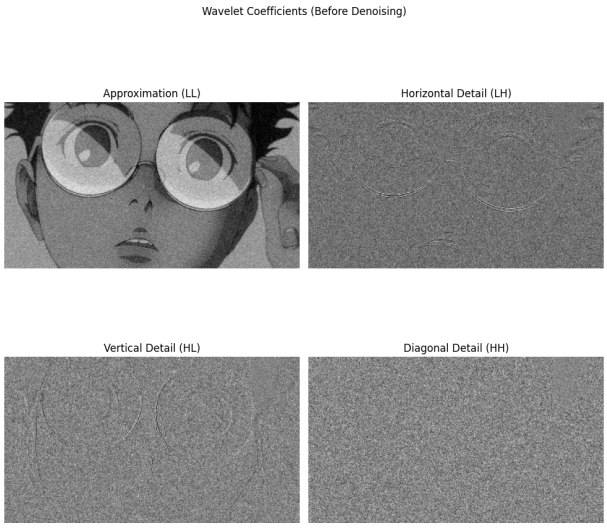
1. **Image Preparation:**
   - Convert the image to grayscale and resize it.
   - Add Gaussian noise ($\mu = 0$, $\sigma = 25$) to simulate a noisy environment.
2. **Wavelet Transformation:**
   - Perform Haar wavelet decomposition using `pywt.dwt2()`.

- Extract the four sub-bands (**LL, LH, HL, HH**).

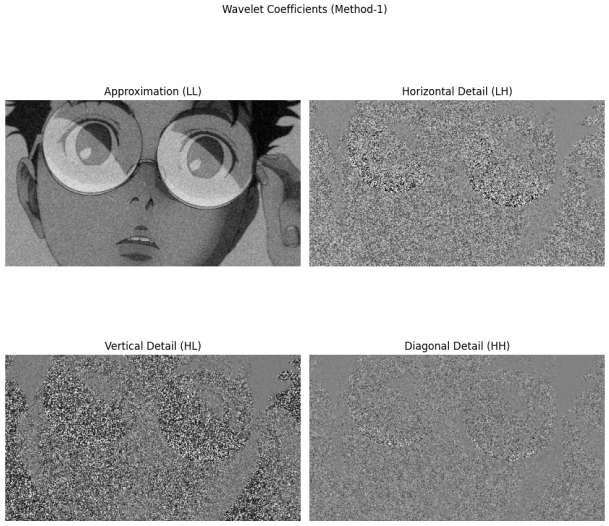
3. **Denoising Techniques:**
   - Method 1: Set all LH, HL, HH coefficients to zero.
   - Method 2: Apply thresholding to remove weak high-frequency components.
4. **Wavelet Reconstruction:**
   - Reconstruct the image using the inverse wavelet transform.
5. **Quality Analysis:**
   - Compute PSNR values to compare the original, noisy, and denoised images.

16.4 Results and Inferences

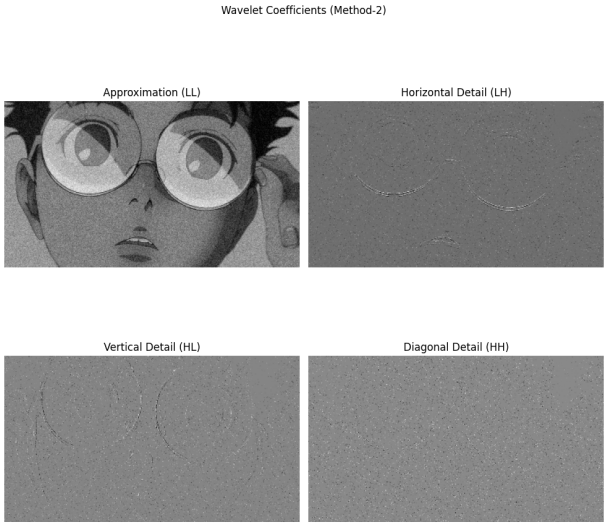
This experiment demonstrates that wavelet-based denoising effectively suppresses noise while preserving structural details. Method-2 (thresholding) provides a better balance between noise reduction and edge preservation, as observed in the PSNR comparison.



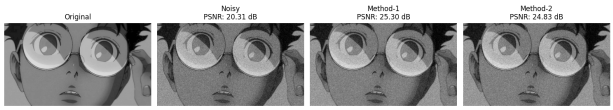
(a) Wavelet Coefficients before Denoising



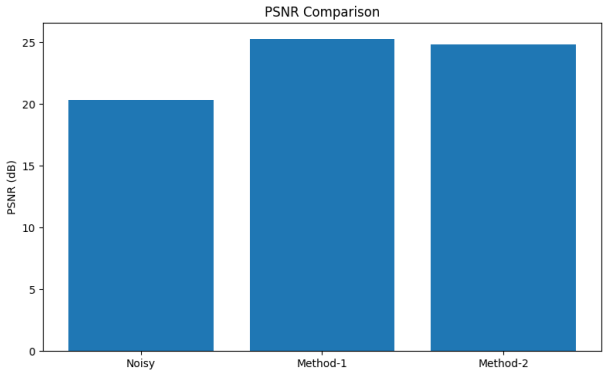
(b) Wavelet Coefficients after Method-1 Denoising



(c) Wavelet Coefficients after Method-2 Denoising



(d) Original vs. Denoised Image



(e) PSNR Comparison of Different Methods

Figure 16: Wavelet-Based Image Denoising Results