

# Digital Image Processing Assignment 1: Image Reading, Enhancement and Resampling

Ian Tsai 313553058  
Department of Computer Science  
National Yang Ming Chiao Tung University  
Hsinchu, Taiwan  
Email: iantsai.cs13@nycu.edu.tw

**Abstract**—This report presents the implementation and results of three fundamental digital image processing techniques: image reading, image enhancement through point operations, and image resampling using interpolation methods. The implementation successfully reads RAW and BMP format images, applies log transform, gamma correction, and negative transform for enhancement, and performs image resizing using nearest neighbor and bilinear interpolation. Experimental results demonstrate the effectiveness of each technique with visual comparisons and quantitative analysis.

**Index Terms**—Digital Image Processing, Point Operations, Image Enhancement, Interpolation, Resampling

**Source Code:** <https://github.com/IaTsai/MMIP-HW1.git>

## I. INTRODUCTION

Digital image processing is a fundamental field in computer vision and multimedia applications. This assignment focuses on implementing basic image processing operations that form the foundation for more complex algorithms. The complete implementation is publicly available at <https://github.com/IaTsai/MMIP-HW1.git>. The objectives include understanding different image formats, implementing enhancement techniques through point operations, and realizing image resampling methods.

The implementation covers three main areas:

- Image reading from RAW and standard formats
- Point-based enhancement operations
- Image resampling with different interpolation methods

## II. METHODOLOGY

### A. Image Reading

The image reading module handles two types of formats:

1) *RAW Format*: RAW images contain uncompressed pixel data in row-major order. The implementation reads binary data and reshapes it into a 512×512 grayscale matrix:

$$I[i, j] = \text{raw\_data}[i \times \text{width} + j] \quad (1)$$

2) *BMP Format*: BMP images are read using the PIL library, converting them to grayscale if necessary. The centered 10×10 pixel values are extracted for verification purposes.

### B. Image Enhancement

Three point operations are implemented for image enhancement:

1) *Log Transform*: The logarithmic transformation enhances dark regions:

$$s = c \cdot \log(1 + r) \quad (2)$$

where  $r$  is the input pixel value normalized to  $[0, 1]$ ,  $s$  is the output, and  $c$  is a constant.

2) *Gamma Transform*: Power-law transformation adjusts overall brightness:

$$s = r^\gamma \quad (3)$$

Different  $\gamma$  values produce different effects:

- $\gamma < 1$ : Brightens the image
- $\gamma > 1$ : Darkens the image
- $\gamma = 2.2$ : Standard monitor correction

3) *Image Negative*: The negative transformation inverts pixel values:

$$s = L - 1 - r \quad (4)$$

where  $L$  is the maximum gray level (256 for 8-bit images).

### C. Image Resampling

Two interpolation methods are implemented for image resizing:

1) *Nearest Neighbor Interpolation*: The simplest method assigns the nearest pixel value:

$$I'(x', y') = I(\lfloor x \rfloor, \lfloor y \rfloor) \quad (5)$$

where  $(x, y)$  are the corresponding source coordinates.

2) *Bilinear Interpolation*: This method uses weighted average of four neighboring pixels:

$$\begin{aligned} I'(x', y') = & (1 - dx)(1 - dy)I(x_1, y_1) \\ & + dx(1 - dy)I(x_2, y_1) \\ & + (1 - dx)dyI(x_1, y_2) \\ & + dx \cdot dy \cdot I(x_2, y_2) \end{aligned} \quad (6)$$

where  $dx = x - x_1$  and  $dy = y - y_1$  are the fractional parts.

## III. IMPLEMENTATION DETAILS

### A. Software Architecture

The implementation uses Python with NumPy for numerical operations, PIL for image I/O, and Matplotlib for visualization. The complete implementation is available as open source [6]. The system has been designed with two architectural approaches:

1) *Original Monolithic Design*: The initial implementation (`hw1_image_processing.py`) contains all functionality in a single `ImageProcessor` class, providing a straightforward approach for the assignment requirements.

2) *Modular Architecture (Extended Implementation)*: To facilitate future expansion and integration with subsequent assignments, a modular architecture has been developed:

- **Core Module** (`core/`): Contains the `ImageProcessor` class with all image processing algorithms
- **Utils Module** (`utils/`): Provides reusable utilities
  - `FileIO`: Unified file reading/writing interface supporting multiple formats
  - `Visualizer`: Enhanced visualization tools with histogram and comparison functions
- **Main Interface** (`main.py`): Command-line interface with argument parsing for flexible execution

This modular design offers several advantages:

- 1) **Reusability**: Core functions can be imported and used in future assignments
- 2) **Maintainability**: Separation of concerns makes code easier to understand and modify
- 3) **Extensibility**: New processing modules can be added without modifying existing code
- 4) **Testing**: Individual modules can be tested independently

#### B. Key Implementation Challenges

1) *Numerical Stability*: Log transform requires adding 1 to avoid  $\log(0)$ . All operations normalize pixel values to  $[0,1]$  before processing to maintain numerical stability.

2) *Boundary Handling*: Interpolation near image borders requires careful index checking to prevent out-of-bounds access.

3) *Performance Optimization*: While Python loops are slower than vectorized operations, the implementation prioritizes clarity over performance for educational purposes.

### IV. EXPERIMENTAL RESULTS

#### A. Test Images

Six test images were used:

- RAW format: lena, goldhill, peppers (512×512, 8-bit grayscale)
- BMP format: boat, baboon, F16 (512×512, converted to grayscale)

#### B. Image Reading Results

All images were successfully loaded and displayed. Table I shows sample center pixel values for verification.

#### C. Enhancement Results

Figure 1 shows the enhancement results for one test image. Key observations:

- 1) *Log Transform*:
  - Effectively brightens dark regions
  - Compresses dynamic range
  - Useful for images with low contrast in dark areas

TABLE I  
SAMPLE CENTER PIXEL VALUES (5×5)

Image	Center 5×5 Pixel Values				
Lena	195	195	195	192	169
	196	196	189	157	124
	196	187	149	123	127
	180	135	116	117	124
	124	102	115	116	120



Fig. 1. Image enhancement results on Lena image

#### 2) Gamma Transform:

- $\gamma = 0.5$ : Significantly brightens the image
- $\gamma = 1.5$ : Slightly darkens while preserving details
- $\gamma = 2.2$ : Standard display correction

#### 3) Negative Transform:

- Complete inversion of gray levels
- Useful for enhancing white or gray details in dark regions

#### D. Resampling Results

Table II summarizes the resampling experiments:

TABLE II  
RESAMPLING TEST CASES

Test Case	Source Size	Target Size
Downsampling 1	512×512	128×128
Downsampling 2	512×512	32×32
Upsampling 1	32×32	512×512
Non-uniform 1	512×512	1024×512
Non-uniform 2	128×128	256×512

#### 1) Downsampling Results:

- Both methods preserve main features
- Bilinear produces smoother results
- Nearest neighbor shows more aliasing artifacts

#### 2) Upsampling Results:

- Nearest neighbor produces blocky artifacts
- Bilinear creates smoother but blurrier results
- Information loss is irreversible

Figure 2 shows a comparison of the two methods:

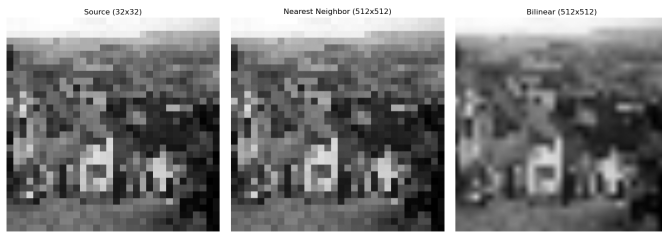


Fig. 2. Comparison of interpolation methods (32×32 to 512×512)

## V. DISCUSSION

### A. Performance Analysis

Bilinear interpolation requires approximately 4× more computations than nearest neighbor but produces significantly better visual quality. The trade-off between quality and speed depends on the application requirements.

### B. Limitations

- Current implementation uses simple loops, which could be optimized with vectorization
- Only grayscale images are supported
- More advanced interpolation methods (bicubic, Lanczos) could improve quality

### C. Potential Improvements and Achievements

1) *Completed Improvements:* The following enhancements have been implemented in the extended version:

- **Modular Architecture:** Successfully refactored into reusable modules
- **Enhanced CLI:** Added command-line arguments for flexible operation
- **Unified I/O Interface:** Created FileIO class for consistent file handling
- **Extended Visualization:** Implemented additional visualization utilities

2) *Future Improvements:*

- Implement vectorized operations for better performance
- Add support for color images
- Include more interpolation methods (bicubic, Lanczos)
- Implement adaptive enhancement based on image statistics
- Develop a graphical user interface (GUI) for interactive processing

## VI. CONCLUSION

This assignment successfully implemented fundamental image processing operations including reading, enhancement, and resampling. The results demonstrate:

- 1) Correct reading of both RAW and standard image formats
- 2) Effective enhancement through point operations, each suitable for different scenarios
- 3) Clear differences between interpolation methods in resampling tasks

### 4) Successful modular architecture design for future extensibility

The implementation provides a solid foundation for understanding digital image processing concepts. The visual and quantitative results confirm the theoretical expectations for each technique. Furthermore, the modular architecture ensures that these implementations can be readily integrated with future assignments, creating a comprehensive image processing system.

The extended implementation with its modular design demonstrates software engineering best practices, including separation of concerns, reusability, and maintainability. This approach not only fulfills the current assignment requirements but also establishes a framework for building more complex image processing applications.

Future work will leverage this modular foundation to incorporate more sophisticated algorithms such as filtering operations, edge detection, morphological processing, and advanced enhancement techniques.

## ACKNOWLEDGMENT

The author would like to thank the course instructor for providing the assignment materials and test images. The complete source code for this project is available at <https://github.com/IaTsai/MMIP-HW1.git>.

## REFERENCES

- [1] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. Pearson, 2018.
- [2] Wikipedia contributors, "Bilinear interpolation," *Wikipedia, The Free Encyclopedia*. [Online]. Available: [https://en.wikipedia.org/wiki/Bilinear\\_interpolation](https://en.wikipedia.org/wiki/Bilinear_interpolation)
- [3] Wikipedia contributors, "Nearest-neighbor interpolation," *Wikipedia, The Free Encyclopedia*. [Online]. Available: [https://en.wikipedia.org/wiki/Nearest-neighbor\\_interpolation](https://en.wikipedia.org/wiki/Nearest-neighbor_interpolation)
- [4] NumPy Developers, "NumPy: The fundamental package for scientific computing with Python," 2023. [Online]. Available: <https://numpy.org/>
- [5] F. Lundh and Contributors, "Python Imaging Library (PIL)," 2023. [Online]. Available: <https://python-pillow.org/>
- [6] I. Tsai, "MMIP-HW1: Digital Image Processing Implementation," 2025. [Online]. Available: <https://github.com/IaTsai/MMIP-HW1.git>