TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

Faculty of Computer Science

Image Processing Project

Zooming and Shrinking in Digital Images

Student: Razvan-Andrei Bobos
Group: 30435
Year: 3rd Year, Semester 2
Coordinator: Ion Augustin Giosan

Academic Year: 2024 – 2025

1. Introduction

1.1. Topic Context

In our days, resizing images, either by zooming in or shrinking, is a fundamental operation. Accurate image scaling without significant loss of quality is a key challenge.

1.2. Issues to Be Solved

When scaling digital images, bad methods may result in visible issues such as pixelation, blurring, or jagged edges. Therefore, more advanced interpolation techniques must be used to produce smoother and more visually pleasing results. This project addresses the following challenges, using these 3 interpolation tehniques:

- Nearest Neighbor Interpolation
- Bilinear Interpolation
- Bicubic Interpolation

1.3. Proposed Objectives

The main objectives of this project are:

- To study and implement three interpolation techniques for image resizing: Nearest Neighbor, Bilinear, and Bicubic.
- To apply these algorithms in C++ using OpenCV and compare the output quality.

1.4. Documentation Structure

This document is structured as follows:

- **Section 1: Introduction** – A brief preview of the chosen theme for the project.
- **Section 2: Theoretical Background** – A study of interpolation methods and their mathematical foundations.
- **Section 3: Design and Implementation** – Description of the implementation steps in OpenCV, code structure.
- **Section 4: Experimental Results** – Comparison of the three interpolation methods with visual and performance analysis.
- **Section 5: Conclusions** – Summary of achievements and the degree of accomplishing the objectives, project evaluation, and future improvements.
- **Section 6: Bibliography** – References and materials used in the documentation.

2. Theoretical Background

2.1. Bibliographical Study from the Literature

The starting point for this project was the textbook by Rafael C. Gonzalez and Richard E. Woods, which presents a description of interpolation techniques used in image processing. Based on their work, I selected three commonly used methods to explore in this project: *nearest neighbor*, *bilinear*, and *bicubic interpolation*.

To better understand how these techniques are implemented and used in practice, I then watched YouTube videos that explained these concepts.

- *Nearest Neighbor Explained* - `https://www.youtube.com/watch?v=ZVuRUXdIwDA`
- *Bilinear Interpolation Explained* – `https://www.youtube.com/watch?v=NnksKpJZEkA`
- *Bicubic Interpolation* – `https://www.youtube.com/watch?v=NnksKpJZEkA`

In addition to the videos, I read articles on Medium and AI Learner.

2.2. Description of the Methods That Can Be Applied

Three common interpolation techniques used for image scaling are:

- **Nearest Neighbor Interpolation**: This method assigns the value of the closest pixel from the original image to the new pixel.
- **Bilinear Interpolation**: This method takes a weighted average of the four closest pixels to estimate the new pixel value.
- **Bicubic Interpolation**: This more complex method uses 16 surrounding pixels and a cubic convolution function to estimate the pixel value.

2.3. Description of the Possible Solutions

In this project, we consider the implementation and comparison of the three interpolation methods mentioned above using C++ and OpenCV. Each method has trade-offs:

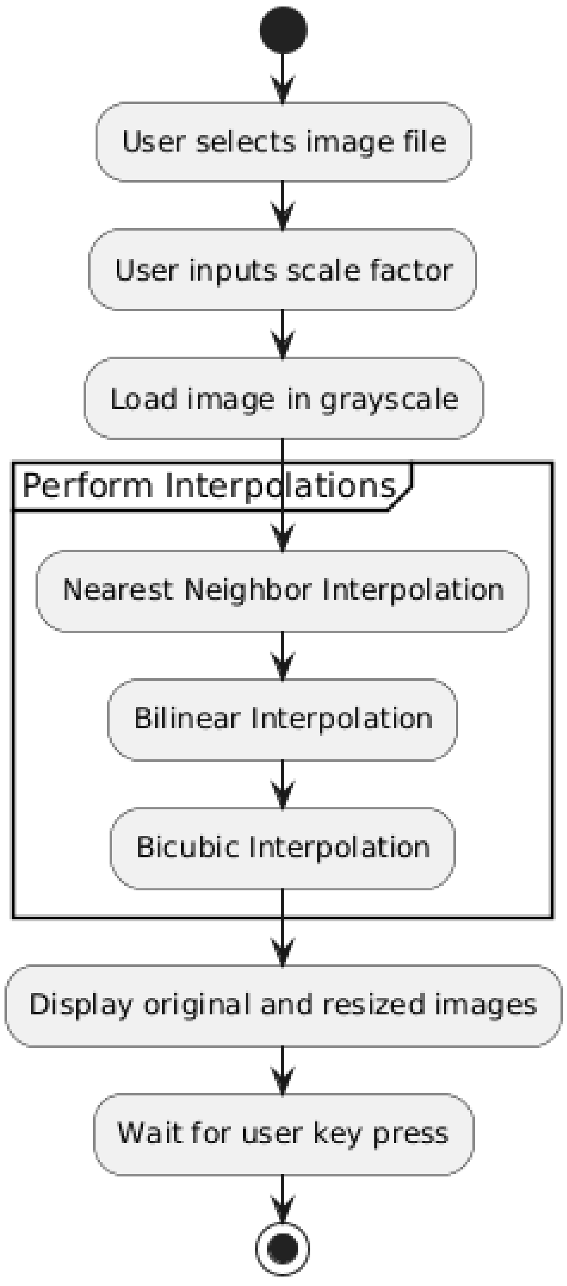
- **Nearest Neighbor** – fast and easy to implement, but produces blocky, pixelated images.
- **Bilinear** – balanced in speed and quality, provides smoother results than nearest neighbor. It produces slight blurring, especially at the edges.
- **Bicubic** – best visual quality, with less blurring. However, it has a slower computation.

3. Design and Implementation

3.1. Description of the Chosen Solution

The goal of this project is to implement and compare three widely-used image interpolation techniques: **nearest neighbor**, **bilinear**, and **bicubic interpolation**. These methods are applied to grayscale images for both zooming and shrinking purposes. The implementation was done in C++ using the OpenCV library. The user is prompted to choose a scaling factor, after which the program resizes the image using all three techniques and displays the results.

3.2. Block Diagram



3.3. Implementation Flow

1. User selects an image using the file dialog.
2. The image is loaded in grayscale format.
3. The user is prompted to enter a scaling factor (e.g., 0.5 for shrinking or 2.0 for zooming).
4. The program calculates the new dimensions and creates three empty destination matrices.
5. It then applies:
   - Nearest neighbor interpolation
   - Bilinear interpolation
   - Bicubic interpolation
6. Each method fills its respective matrix using the algorithm logic.
7. All three resized images are displayed alongside the original.

3.4. Description of Functionalities

- `openFileDlg(fname)` – Opens the file dialog to select an image.
- `imread(fname, IMREAD_GRAYSCALE)` – Reads the image in grayscale.
- `Mat dstX` – Creates destination matrices for the three resized images.
- `imshow(...)` – Displays the resulting images using OpenCV.
- `waitKey()` – Waits for a key press to close the display windows.

3.5. Description of Implemented Algorithms

Nearest Neighbor

This method assigns to each output pixel the value of the nearest input pixel based on rounding down the floating-point coordinates. It's fast but may result in jagged edges and low-quality scaling.

Bilinear Interpolation

This technique considers the four nearest neighbors of a pixel and performs linear interpolation first along the X axis and then along the Y axis. The resulting value is smoother than nearest neighbor and performs well for moderate scaling.

Bicubic Interpolation

This is the most complex of the three methods and provides the best visual quality. The algorithm considers a 4×4 neighborhood and uses cubic convolution functions in both directions. The interpolation weights are calculated using the following function:

$$f(s) = \begin{cases} (a+2)|s|^3 - (a+3)|s|^2 + 1, & \text{if } |s| \leq 1 \\ a|s|^3 - 5a|s|^2 + 8a|s| - 4a, & \text{if } 1 < |s| \leq 2 \\ 0, & \text{otherwise} \end{cases}$$

In this implementation, the coefficient $a = 0.5$, which is a commonly used value in image processing software such as Adobe Photoshop.

3.6. Application Usage

After launching the application, the user can navigate a menu. By selecting option **13 – Zooming & Shrinking in Digital Images**, they can open an image file. The user is then prompted to input a scaling factor (e.g., 2 for doubling the size or 0.5 for reducing it by half). The application will then display four images:

- The original grayscale image.
- The resized image using nearest neighbor interpolation.
- The resized image using bilinear interpolation.
- The resized image using bicubic interpolation.

Each method visually demonstrates how different interpolation methods affect the image's appearance during resizing.

4. Experimental Results

The images were processed using Nearest Neighbor, Bilinear, and Bicubic interpolation methods. I also present the observations for each scenario, with comments on the expected outcomes.

4.1. Scenario 1: Photograph with Smooth Gradients (Landscape)

Images Tested: HawkesBayesNZ.bmp (1024x683 pixels)
Observations:

- Nearest Neighbor will likely create pixelated or blocky transitions, especially in smooth regions.
- Bilinear will show smoother transitions, but may lose some detail, especially at edges.
- Bicubic will likely provide the smoothest transitions with more accurate color blending, though it may take more time to process.

To be honest, the difference between bilinear and bicubic is not very noticeble in this example.

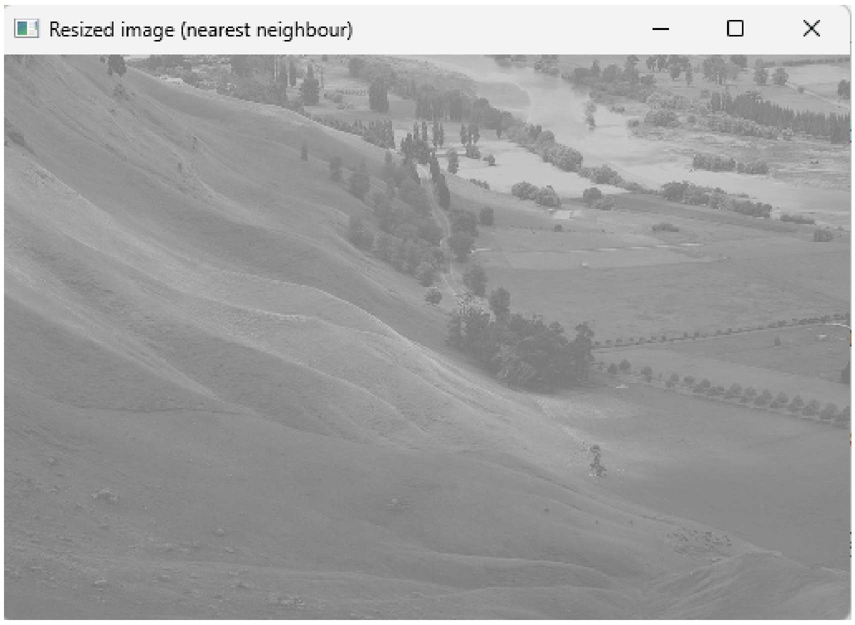


Figure 1: Nearest Neighbor Interpolation

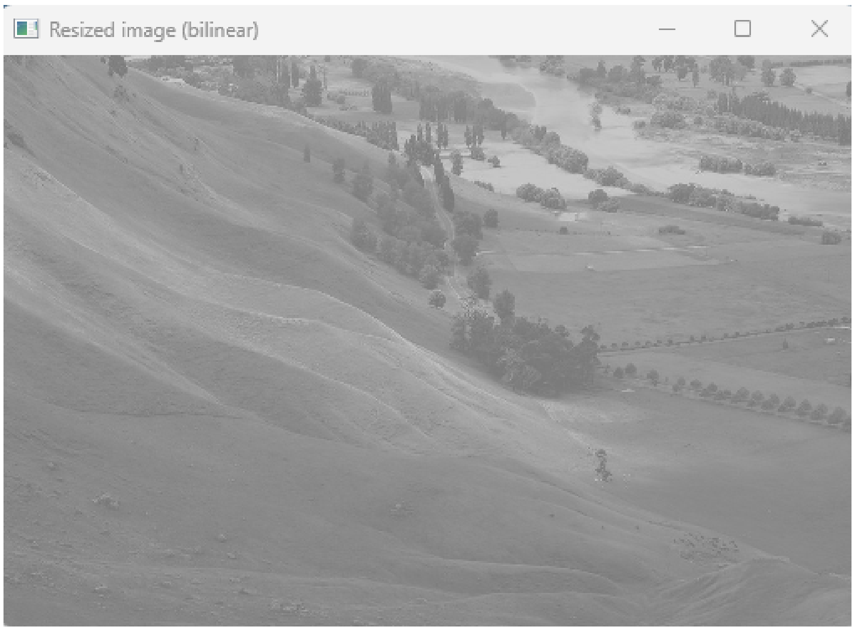


Figure 2: Bilinear Interpolation

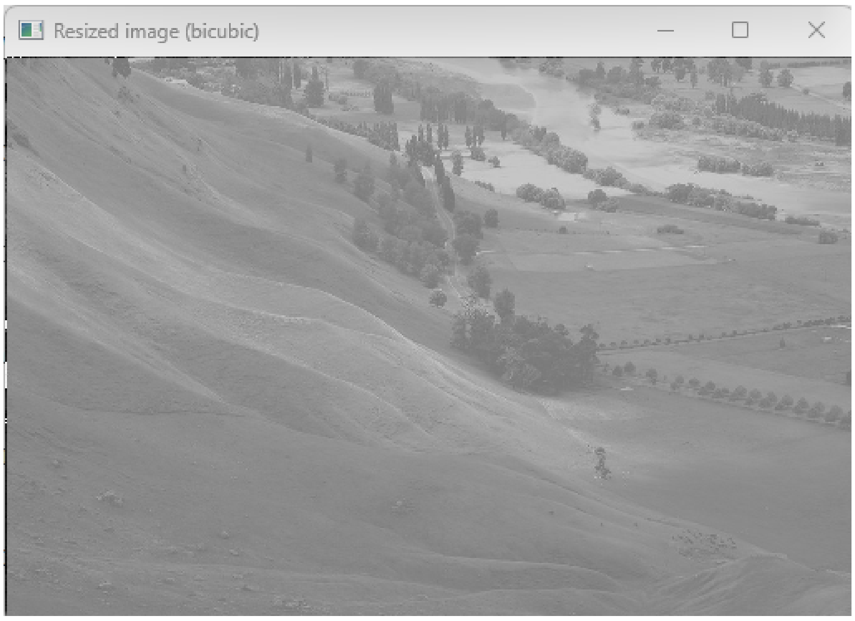


Figure 3: Bicubic Interpolation

4.2. Scenario 2: Images with Sharp Edges

Images Tested: accidentSharpEdges.bmp
Expected Observations:

- Nearest Neighbor should give the best results for sharp edges, preserving the high contrast and clear boundaries.
- Bilinear may cause some blurring of edges, especially for very sharp details.
- Bicubic might cause some oversmoothing, which may lead to blurred text.

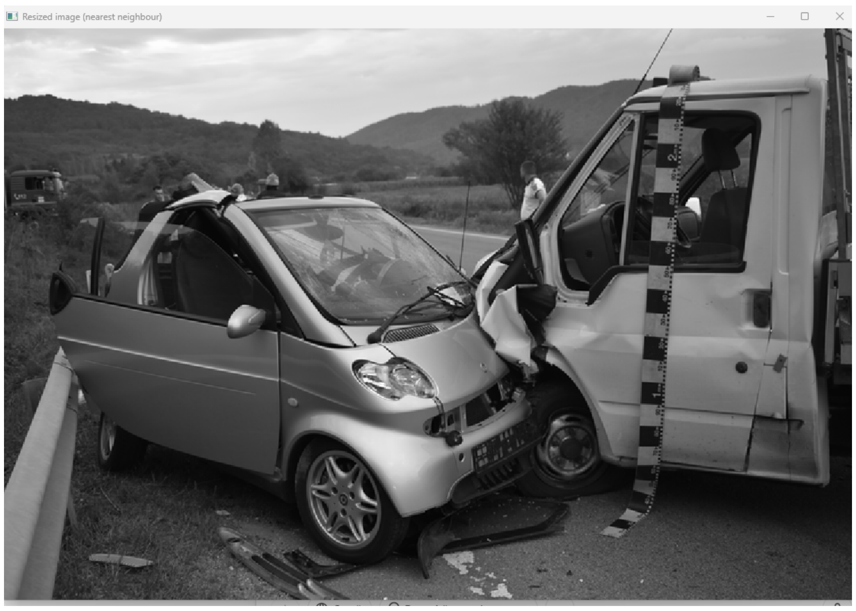


Figure 4: Nearest Neighbor Interpolation

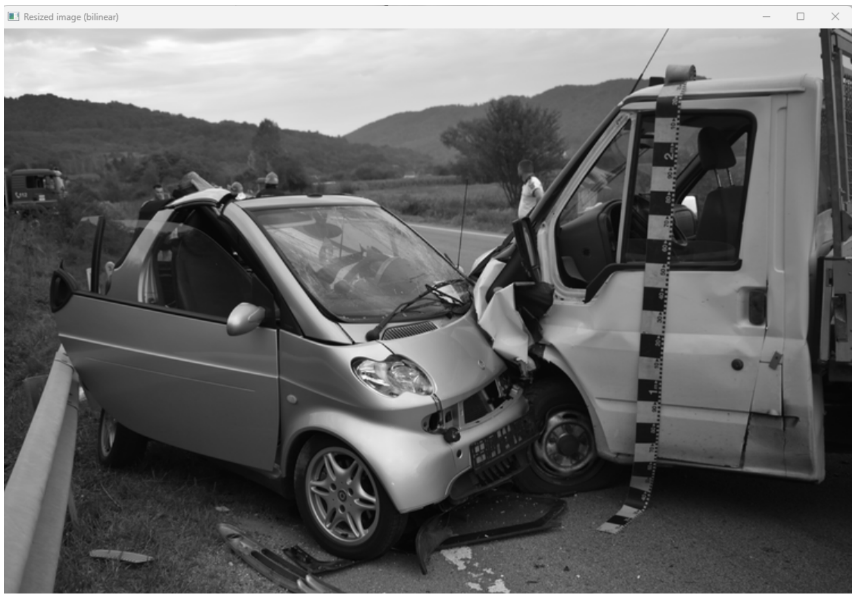


Figure 5: Bilinear Interpolation

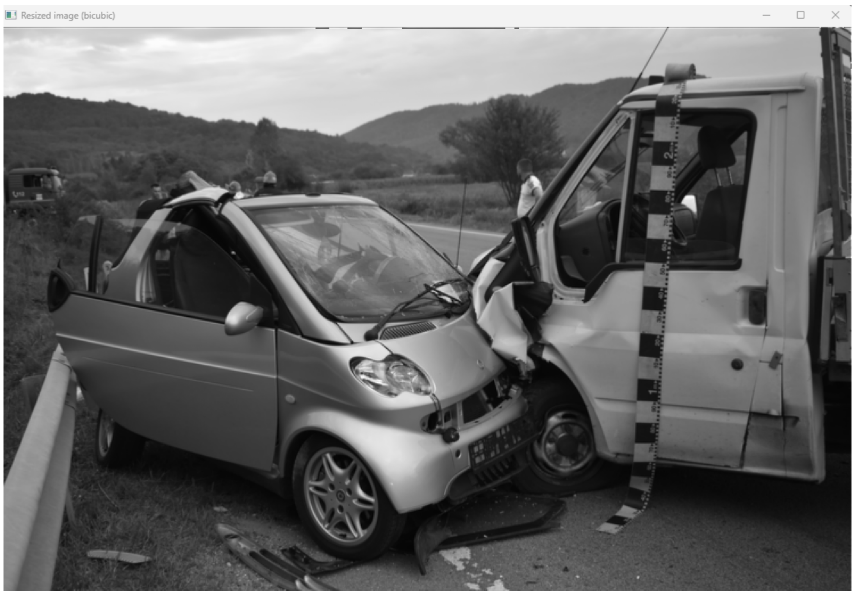


Figure 6: Bicubic Interpolation

4.3. Scenario 3: Black and White High-Contrast Images

Image Tested: phn1thr1bw.bmp
Expected Observations:

- Nearest Neighbor should be ideal for preserving the sharp contrast and blocky structure in these images.
- Bilinear might cause some unwanted blending between adjacent pixels, leading to a loss of sharpness.
- Bicubic may oversmooth the edges and lead to a loss of detail in pixel-based images.

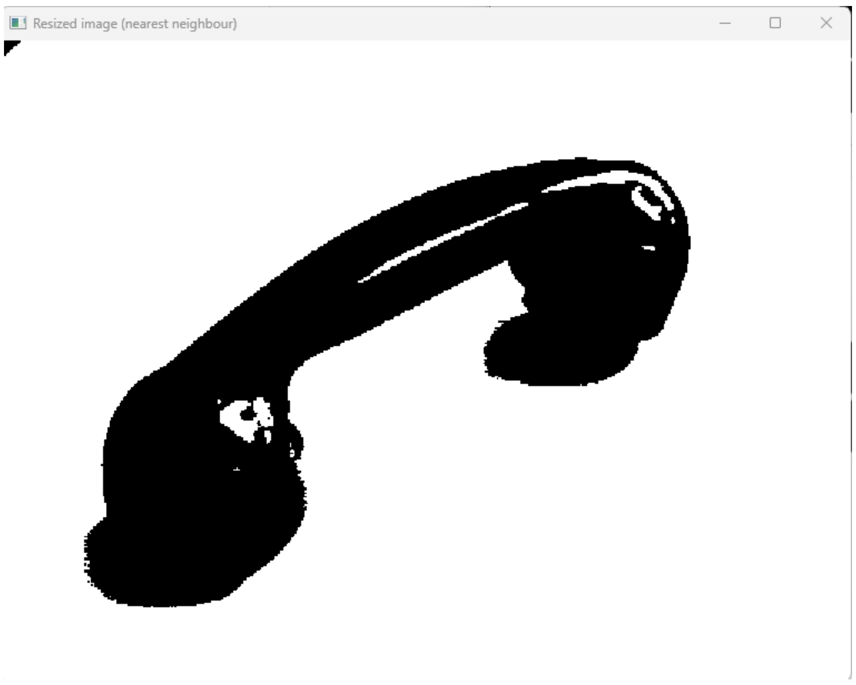


Figure 7: Nearest Neighbor Interpolation

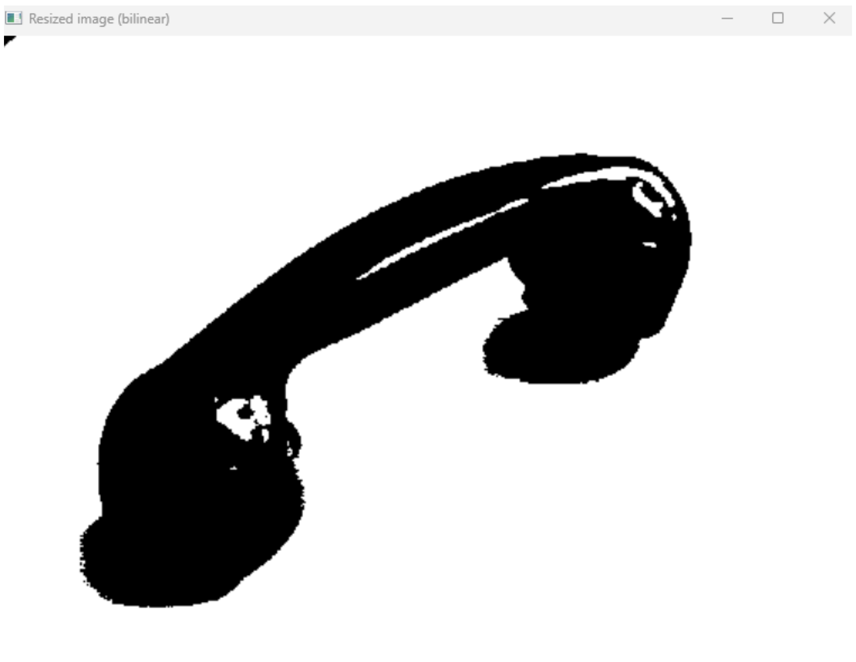


Figure 8: Bilinear Interpolation

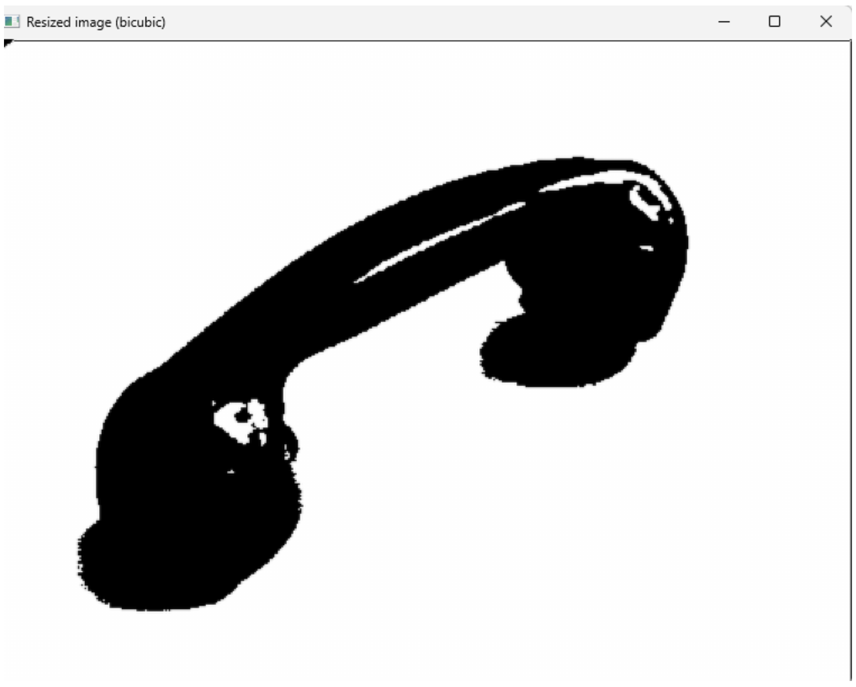


Figure 9: Bicubic Interpolation

4.4. Scenario 4: Noisy or Low-Resolution Images (Simulated Bad Quality Input)

Images Tested: accident pixelata.bmp
Expected Observations:

- Nearest Neighbor might not show significant improvement but will avoid creating blurry results.
- Bilinear will likely smooth out some of the noise, but may blur important features in the image.
- Bicubic will provide a smoother result but may also introduce visible artifacts due to the noise.

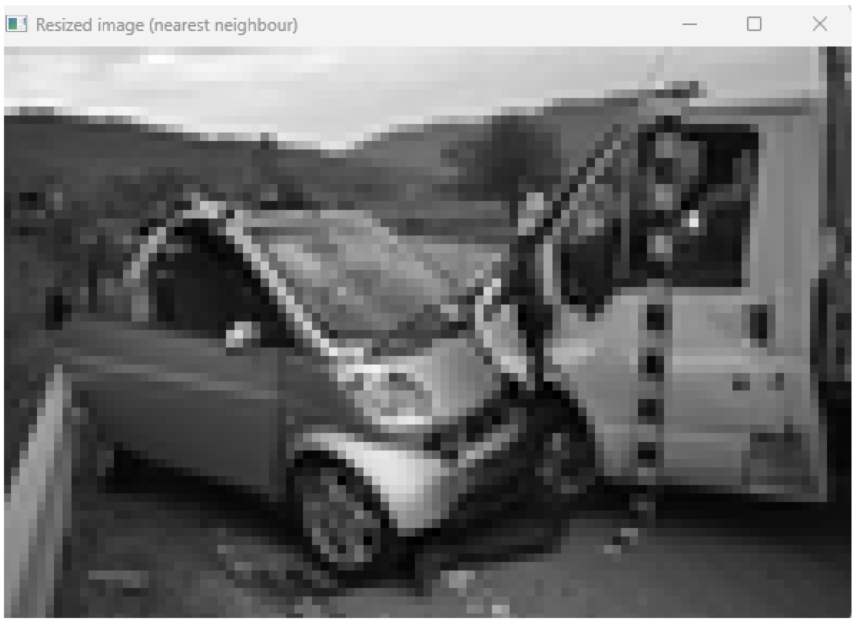


Figure 10: Nearest Neighbor Interpolation

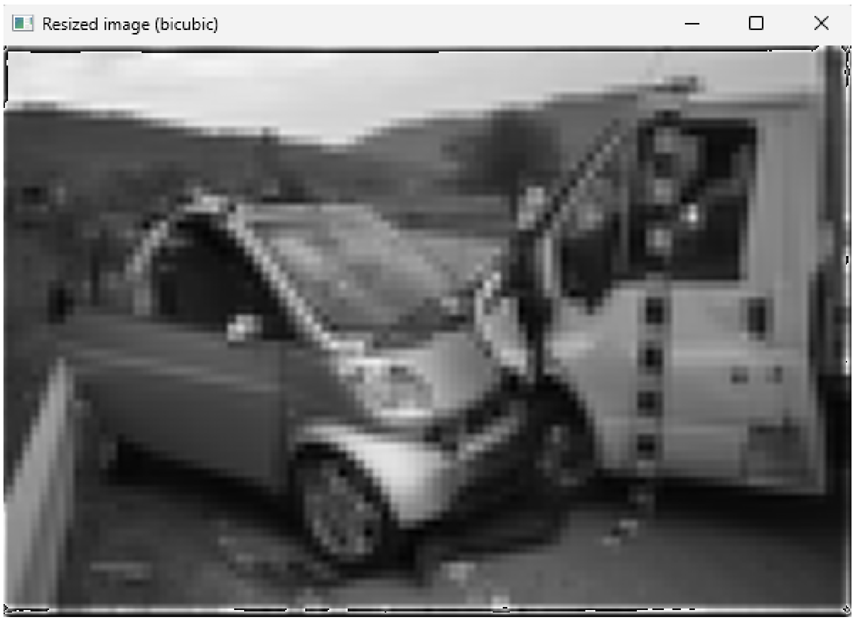


Figure 11: Bilinear Interpolation

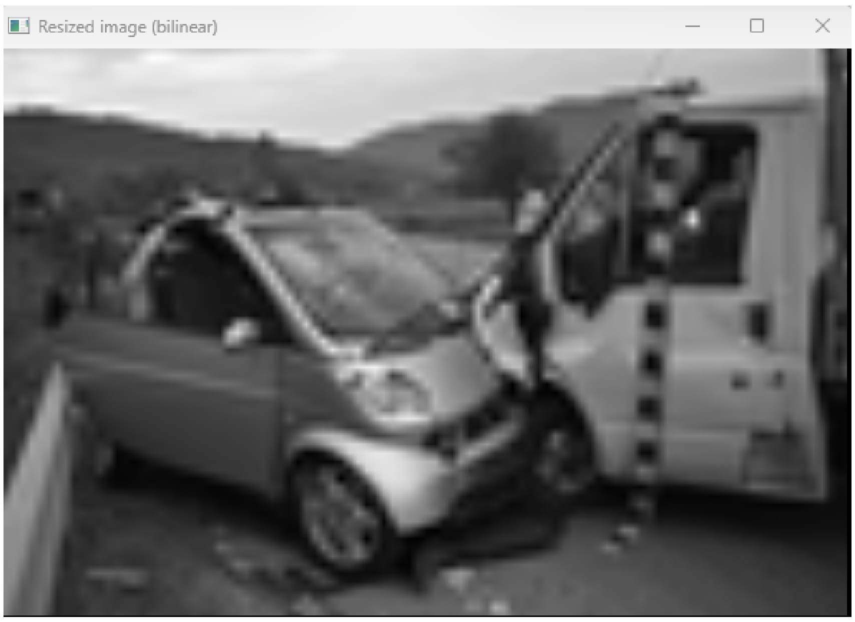


Figure 12: Bicubic Interpolation

5. Conclusions

5.1. Achievements and Accomplishments

The primary objectives of this project were to evaluate the performance of various interpolation techniques (Nearest Neighbor, Bilinear, and Bicubic) on different types of images. The project tested these methods on various image categories, including photographs with smooth gradients, sharp-edged images, high-contrast black-and-white images, and noisy or low-resolution images. The results were consistent with the theoretical expectations, with Bicubic interpolation producing the smoothest results, while Nearest Neighbor preserved sharp edges best.

5.2. Personal Contributions

I was responsible for:

- Implementing and coding the interpolation techniques.
- Testing the methods on different image categories.
- Analyzing the obtained data.
- Writing the documentation.

5.3. Observations on the Obtained Results

The experimental results validated the theoretical expectations for each interpolation technique:

- Nearest Neighbor produced blocky results on smooth images but excelled in preserving sharp edges in images with high contrast.
- Bilinear interpolation provided smoother transitions in most cases, though it occasionally blurred edges and details.
- Bicubic interpolation offered the smoothest results but was computationally more expensive and, in some cases, led to oversmoothing.

5.4. Future Development Directions

There are several potential directions for further development:

- **User Interface Improvements:** Creating a user-friendly graphical interface.
- **Optimization of Computational Efficiency:** Exploring more efficient interpolation techniques.
- **Improvement in Image Quality for Experiments:** Using higher-quality images for testing or come up with better scenarios.

6. Bibliography

Below is a list of references used in the preparation of this documentation:

- **Gonzalez, R.C., Woods, R.E.** (2008). *Digital Image Processing.* Pearson.
- `https://www.canva.com/design/DAGY0iX3k/3kwFpaOM2sAZDV0n0pco2g/edit`
- `https://meghal-darji.medium.com/implementing-bilinear-interpolationfor-image-res`
- `https://theailearner.com/2018/12/29/image-processing-bicubic-interpolation/`
- `https://medium.com/@akp83540/nearest-neighbor-interpolation-84c956ee56a3/`
- `https://users.utcluj.ro/~igiosan/Resources/PI/making_documentation.pdf`