Image resizing.

Abstract

This Python script demonstrates image processing capabilities using the OpenCV library (cv2) and visualization techniques with matplotlib. The script focuses on image resizing operations with various interpolation methods, performance benchmarking, and visual representation of resized images and differences.

The script begins by loading an input image and resizing it using three different interpolation methods (INTER_NEAREST, INTER_LINEAR, INTER_CUBIC) provided by OpenCV. It measures the time taken for 1000 iterations of each resizing operation to evaluate their efficiency. Additionally, a custom image resizing function is implemented for comparative analysis.

After resizing, the script utilizes matplotlib to display the resized images in a 2x2 grid, showcasing the visual differences between the interpolation methods and the custom resizing approach. It further visualizes these differences using 3D surface plots, highlighting variations in grayscale intensity between the original and resized images.

Objective

The objective of this Python script is to demonstrate proficient use of the OpenCV library (cv2) and matplotlib for image processing tasks. Specifically, the script aims to:

1. **Load and Display Images:** Load an input image and display it using matplotlib for visualization.
2. **Resize Images:** Utilize OpenCV's cv2.resize() function to resize the loaded image using three different interpolation methods (INTER_NEAREST, INTER_LINEAR, INTER_CUBIC).
3. **Measure Performance:** Measure the execution time for 1000 iterations of each interpolation method to assess their efficiency in image resizing tasks.
4. **Custom Resizing:** Implement a custom image resizing function as an alternative approach to resizing.
5. **Visualize Results:** Display the resized images side by side for visual comparison using matplotlib subplots. Additionally, visualize differences between the original and resized images through 3D surface plots.
6. **Error Handling:** Implement robust error handling to manage exceptions, such as FileNotFoundError, ensuring reliable image loading and processing.
7. **Educational Purpose:** Serve as a practical example for developers and researchers to understand and compare different image resizing techniques, emphasizing both performance metrics and visual fidelity.

Problem Statement: Image Resizing Using OpenCV and Custom C/C++ Function

The task involves implementing image resizing using OpenCV's cv::resize function with INTER_NEAREST, INTER_LINEAR, and INTER_CUBIC methods. Measure the time for 1000 iterations per method. Develop and integrate a custom C/C++ function for resizing. Validate its output against OpenCV's results. Evaluate and compare performance, ensuring optimal CPU usage for accurate timing and efficiency. Documentation should include timing results, accuracy of custom function outputs, and insights into performance differences between the custom implementation and OpenCV's standard methods

OPENCV:

OpenCV is a robust open-source library for computer vision and image processing. It supports a wide array of functions for tasks like image manipulation, feature detection, object recognition, and machine learning integration. Optimized for performance across platforms, OpenCV is used in diverse applications including robotics, medical imaging, surveillance, and augmented reality. Its extensive community support and comprehensive documentation make it a preferred choice for developers seeking efficient solutions in computer vision.

- **Image Processing**: OpenCV provides a comprehensive set of functions for image manipulation, processing, and analysis. This includes basic operations like loading, saving, resizing, rotating, and cropping images, as well as more advanced techniques such as image filtering, edge detection, and morphological operations.

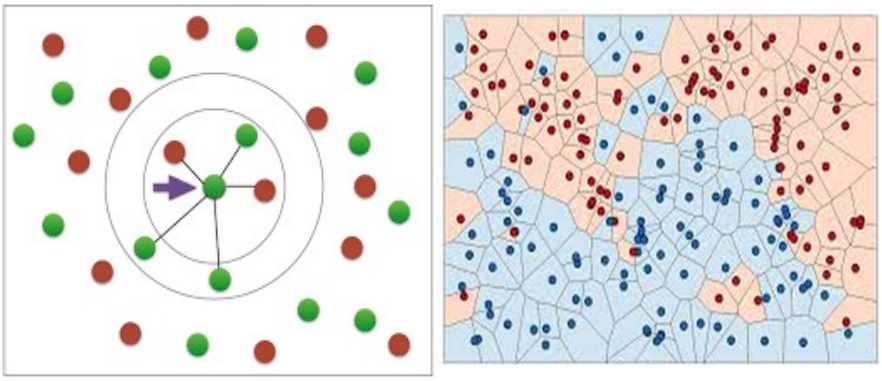
- **Computer Vision Algorithms**: It offers implementations of numerous computer vision algorithms, including feature detection (e.g., Harris Corner Detection, SIFT, SURF), object detection and recognition (e.g., Haar cascades, HOG), optical flow, stereo vision, and camera calibration.

Interpolation:

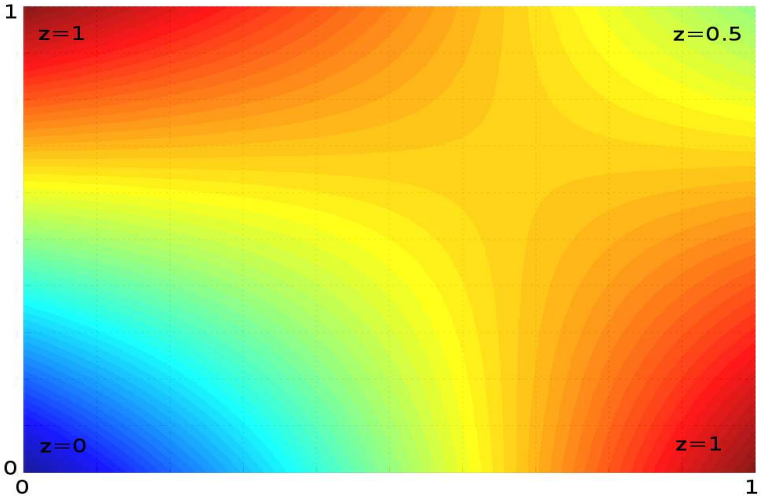
Interpolation in the context of image processing refers to the technique of estimating new data points within the range of known data points. In the case of image resizing, interpolation is used to determine the color or intensity values of new pixels that are added or removed when resizing an image to a different size.

There are several interpolation methods commonly used in image processing:

1. **Nearest Neighbor**: The simplest method where each new pixel in the resized image is assigned the value of the nearest pixel from the original image. It's fast but can result in jagged edges and loss of detail.



2. **Bilinear**: A more sophisticated method where each new pixel is calculated as a weighted average of the four nearest pixels in the original image. It smooths edges and produces better quality than nearest neighbor.

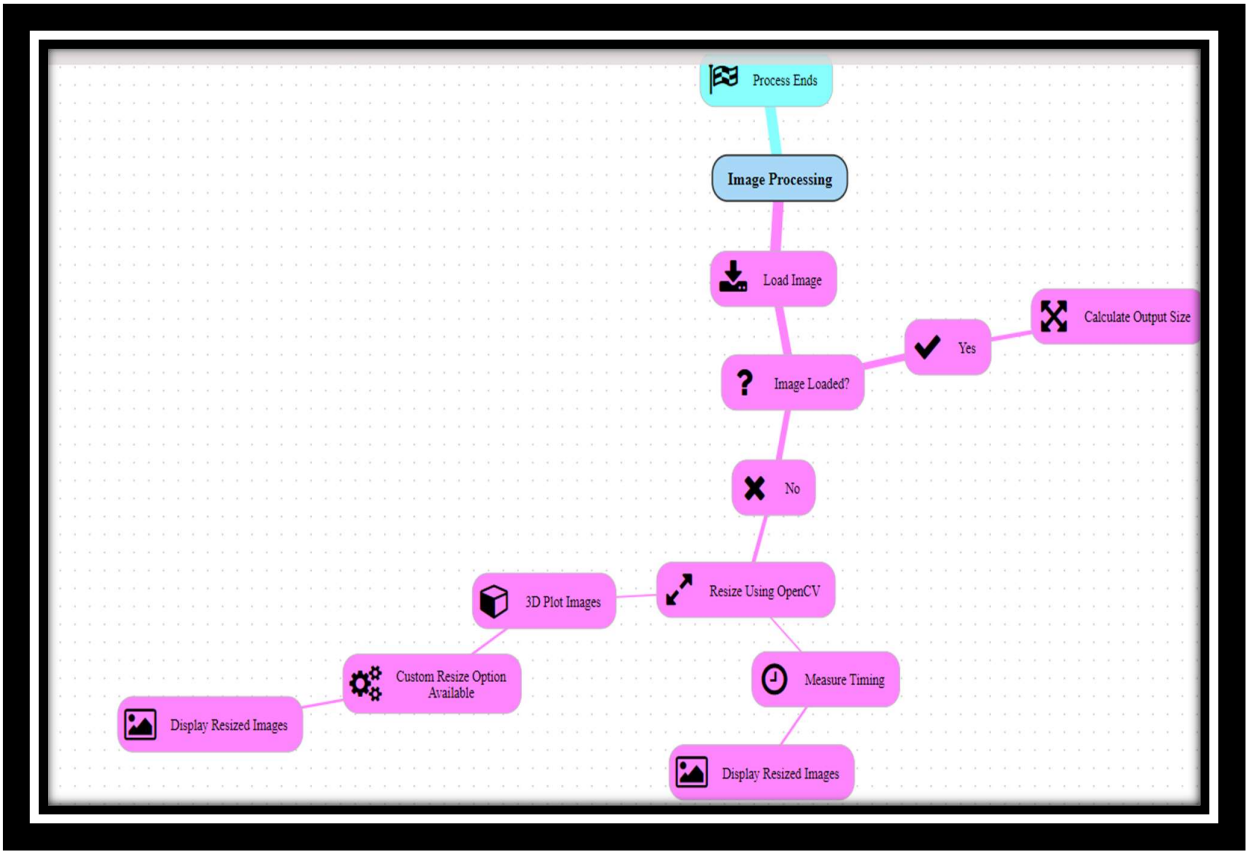


3. **Bicubic**: A higher-order interpolation method that considers a larger neighborhood of pixels (typically 16) to compute the new pixel value. It offers smoother results and is often preferred for high-quality image resizing.

Interpolation plays a crucial role in maintaining image quality when resizing, as it determines how accurately the new pixels represent the original image content. The choice of interpolation method depends on factors such as the desired output quality, computational resources, and application requirements.

Versions:

Package	Version
contourpy	1.1.1
cycler	0.12.1
fonttools	4.53.0
importlib_resources	6.4.0
kiwisolver	1.4.5
matplotlib	3.7.5
numpy	1.24.4
opencv-python	4.10.0.84
packaging	24.1
pillow	10.3.0
pip	23.2.1
pyparsing	3.1.2
python-dateutil	2.9.0.post0
setuptools	68.2.0
six	1.16.0
wheel	0.41.2
zipp	3.19.2



Process:

1. Loading the Image

The script starts by loading an image from a specified path using the load_image() function. This function uses OpenCV's cv2.imread() to read the image as a NumPy array. If the image cannot be loaded (e.g., file not found), it raises a FileNotFoundError.

2. Resizing Images with Different Interpolation Methods

Once the image is loaded, the script performs resizing using three different interpolation methods: INTER_NEAREST, INTER_LINEAR, and INTER_CUBIC. This is done in the resize_with_interpolation() function.

- **Implementation Details:**
  - **Multithreading:** Uses ThreadPoolExecutor to execute resizing tasks concurrently, leveraging multiple threads for efficiency.
  - **Asynchronous Execution:** Submits resizing tasks as futures (executor.submit()), and collects results as they complete (as_completed()).
  - **Output:** Returns resized images using each interpolation method (INTER_NEAREST, INTER_LINEAR, INTER_CUBIC) as separate outputs.

3. Timing Measurement for Resizing Operations

To evaluate performance, the script measures the time taken for resizing operations using each interpolation method over 1000 iterations. This is implemented in the measure_timing() function.

- **Implementation Details:**
  - **Time Measurement:** Uses time.time() to record start and end times for each resizing operation.
  - **Repeated Resizing:** Resizes the image 1000 times for each interpolation method to compute average time.
  - **Output:** Returns timing results in milliseconds for INTER_NEAREST, INTER_LINEAR, and INTER_CUBIC methods.

4. Custom Resizing Function Implementation

The script includes a custom resizing function, custom_resize(), which implements a simple nearest-neighbor interpolation method for resizing an image. This function manually calculates the mapping from destination pixels to source pixels based on the output size.

- **Implementation Details:**
  - **Nearest-Neighbor Interpolation:** Uses simple calculation to map pixels from original to resized image dimensions.
  - **Output:** Returns the resized image as a NumPy array.

5. Displaying Resized Images

After performing resizing using different methods (both OpenCV's built-in methods and the custom method), the script displays the results visually using matplotlib.

- **Implementation Details:**
  - **Visualization:** Uses matplotlib.pyplot to create a 2x2 grid of subplots.
  - **Display:** Shows the images resized using INTER_NEAREST, INTER_LINEAR, INTER_CUBIC methods, and the custom method.
  - **Color Conversion:** Converts BGR images (used by OpenCV) to RGB format for display.

6. Plotting 3D Images with Differences Highlighted

For a more detailed comparison, the script plots 3D representations of the resized images, highlighting differences between the original and resized images.

- **Implementation Details:**
  - **3D Plotting:** Uses mpl_toolkits.mplot3d to create 3D surface plots.
  - **Difference Calculation:** Converts images to grayscale, calculates absolute differences, and plots them using contourf for visual distinction.
  - **Downsampling:** Downscales images and differences for faster plotting.

7. Example Usage and Integration

Finally, the script demonstrates the entire workflow by incorporating all the above steps in a cohesive manner.

- **Example Workflow:**
  - Loads an example image.
  - Performs resizing using built-in and custom methods.
  - Measures and prints timing results for performance evaluation.
  - Displays resized images for visual comparison.
  - Plots 3D representations with differences highlighted.

Error :

Error: name 'uint8' is not defined.

File Not Found Error Handling

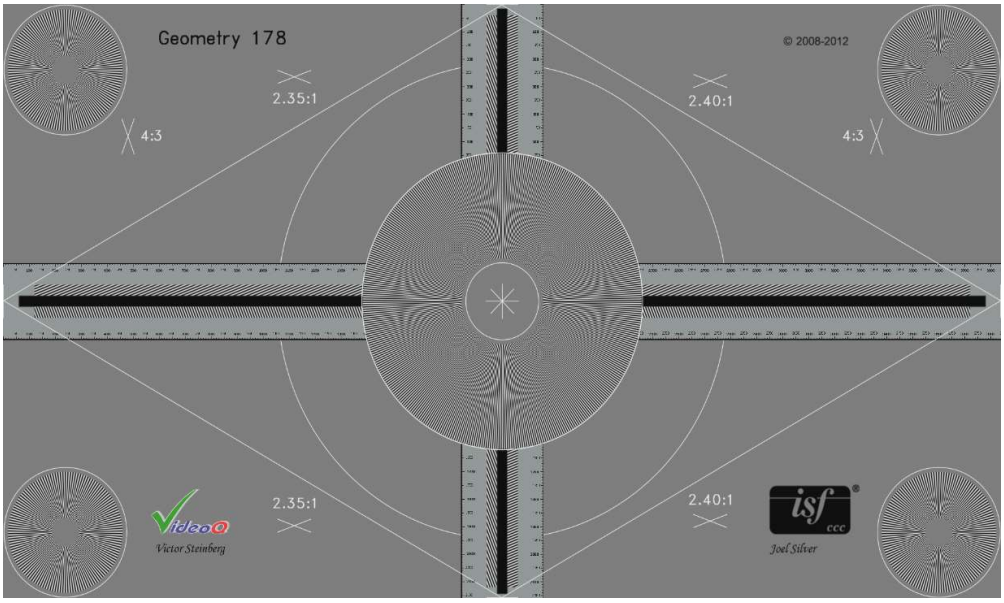
OpenCV Installation Issues

Custom Resize Function Accuracy

Links :

Github: https://github.com/BHANUGANESH342/DeepEdge-AI-/upload/main

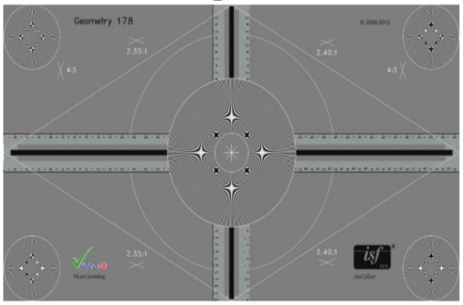
Output images :



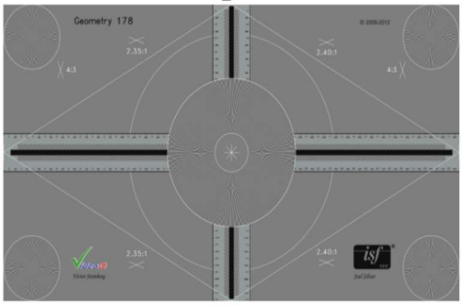
Original image

Resized Images Using Different Interpolation Methods

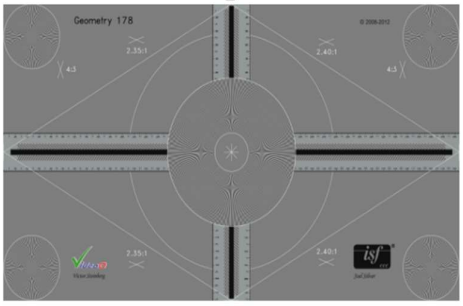
INTER_NEAREST



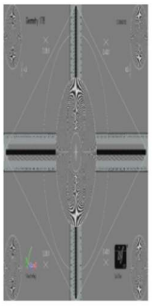
INTER_LINEAR

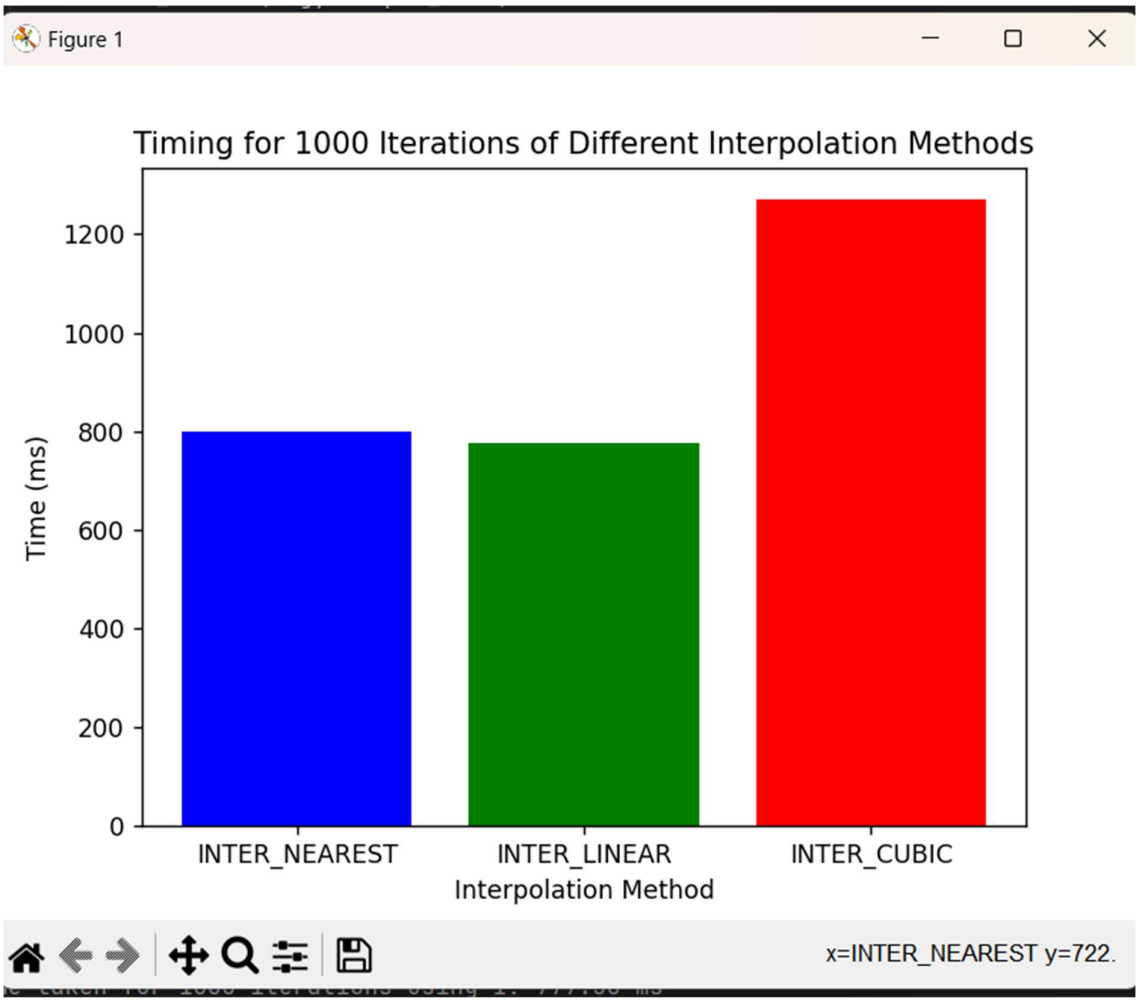


INTER_CUBIC



Custom Resized






```
D:\bhanu\desktop\Projects_working\veipvev\.venv\Scripts\python.exe D:\bhanu\desktop\Projects_working\veipvev\working.py
Time taken for 1000 iterations using 0: 798.62 ms
Time taken for 1000 iterations using 1: 777.50 ms
Time taken for 1000 iterations using 2: 1270.25 ms

Process finished with exit code 0
```

3d images :

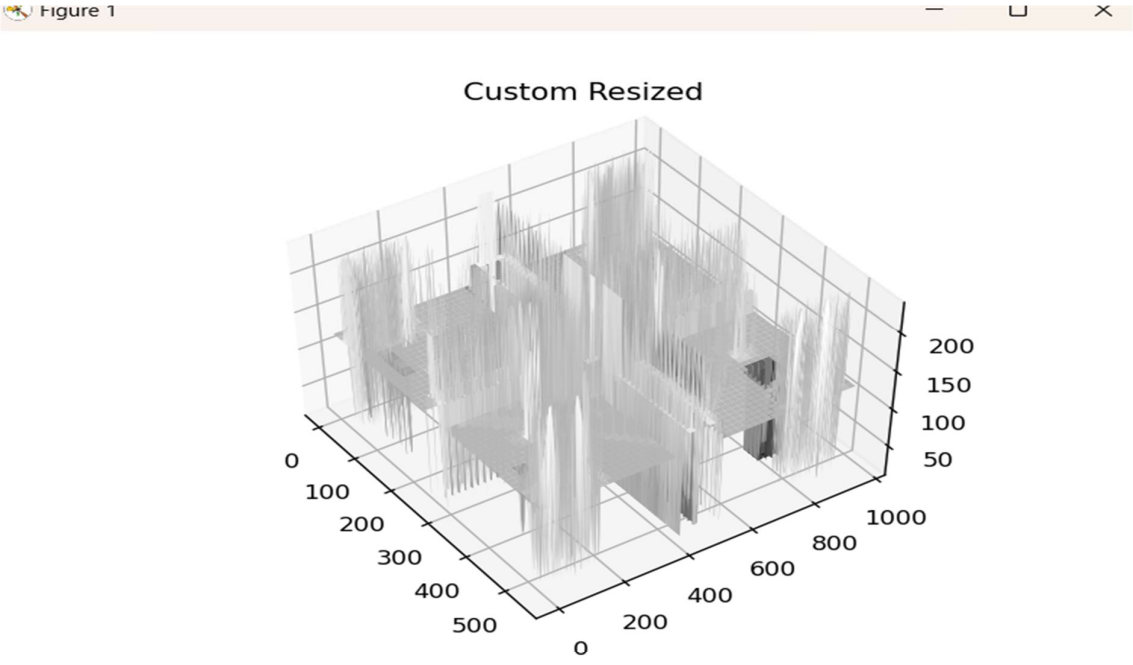
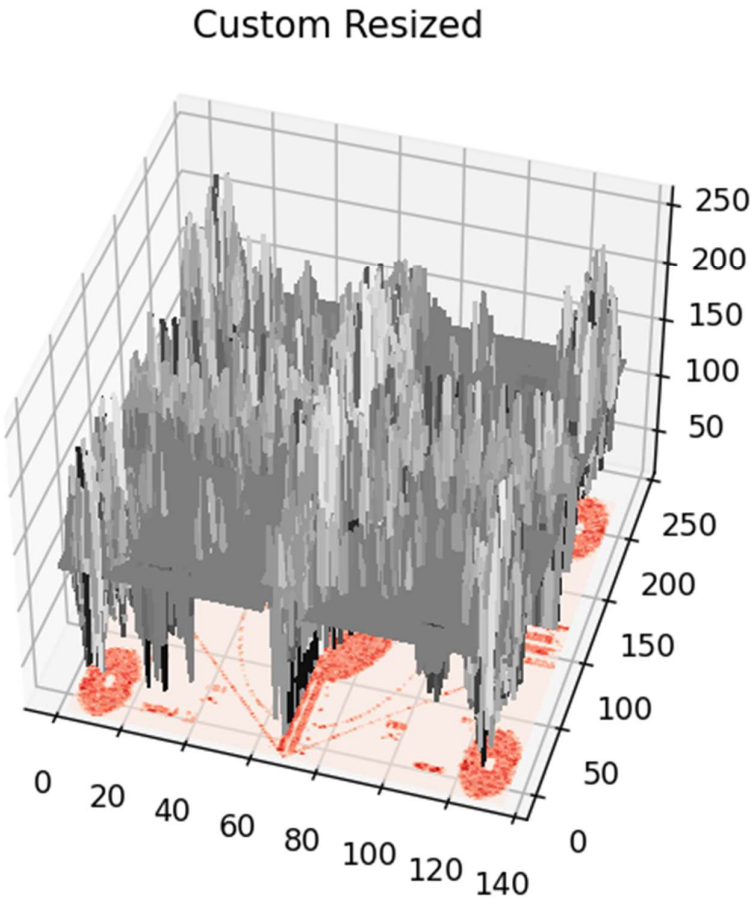


Figure 1
Custom Resized
0
100
200
300
400
500
0
200
400
600
800
1000
200
150
100
50


Custom Resized
0
20
40
60
80
100
120
140
0
50
100
150
200
250
50
100
150
200
250

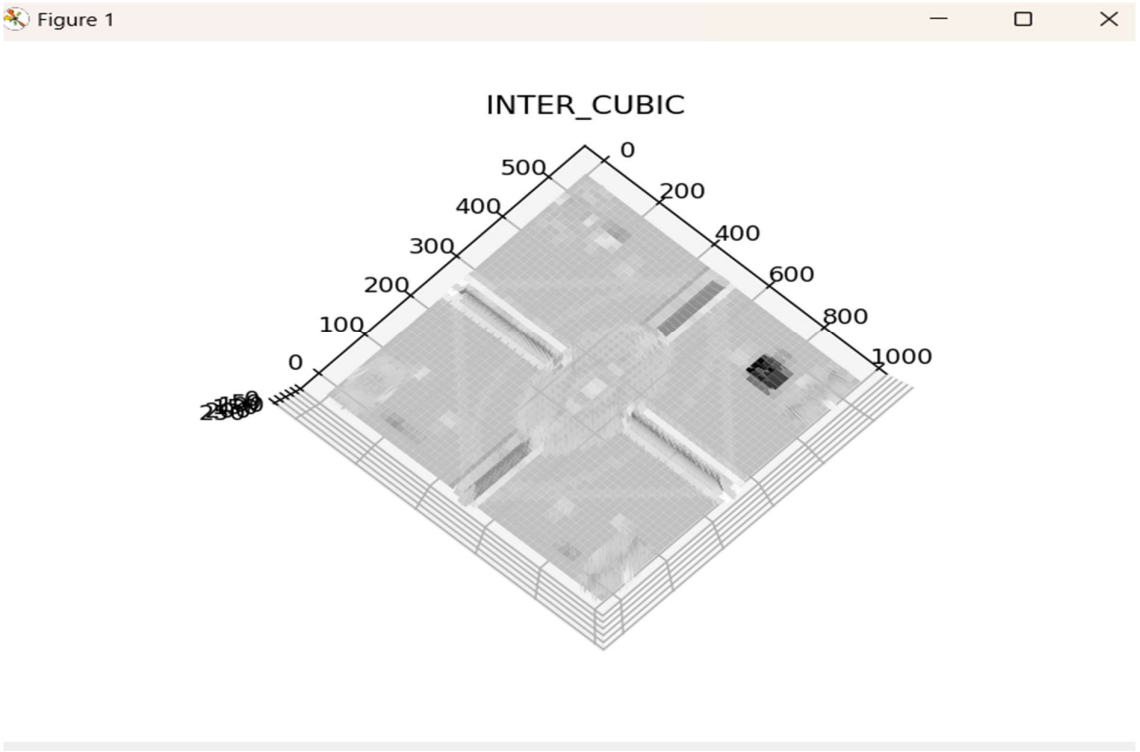
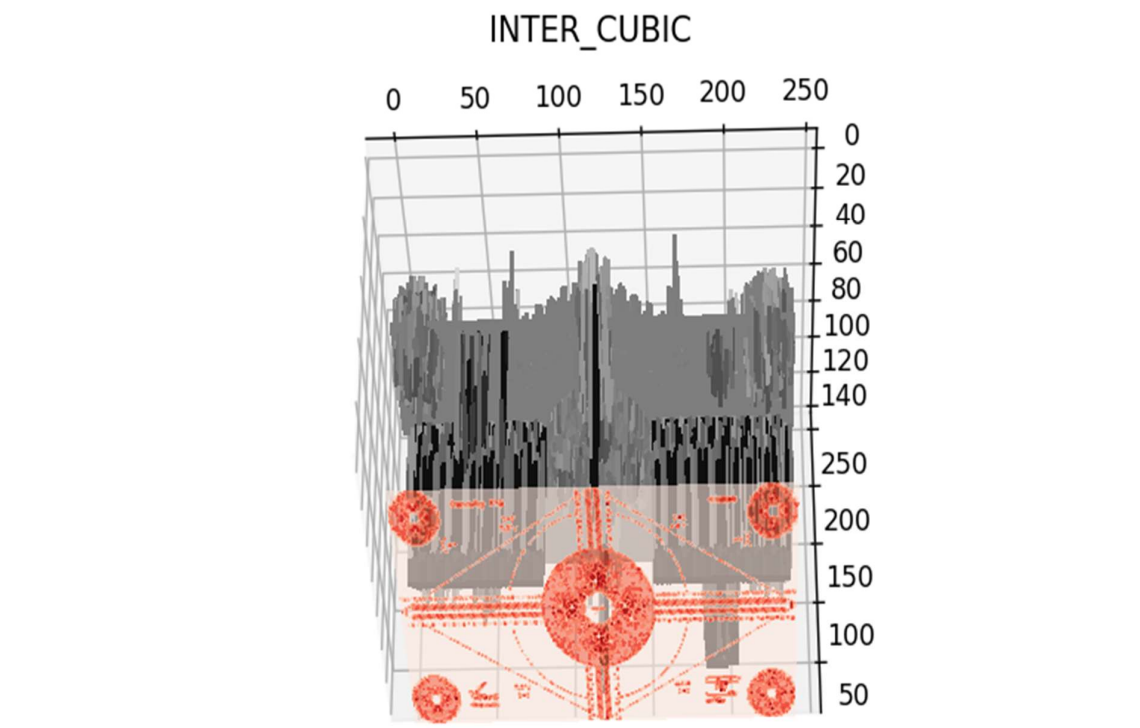


Figure 1
INTER_CUBIC
0
500
200
400
400
300
600
200
800
100
1000
0


INTER_CUBIC
0
50
100
150
200
250
0
20
40
60
80
100
120
140
250
200
150
100
50

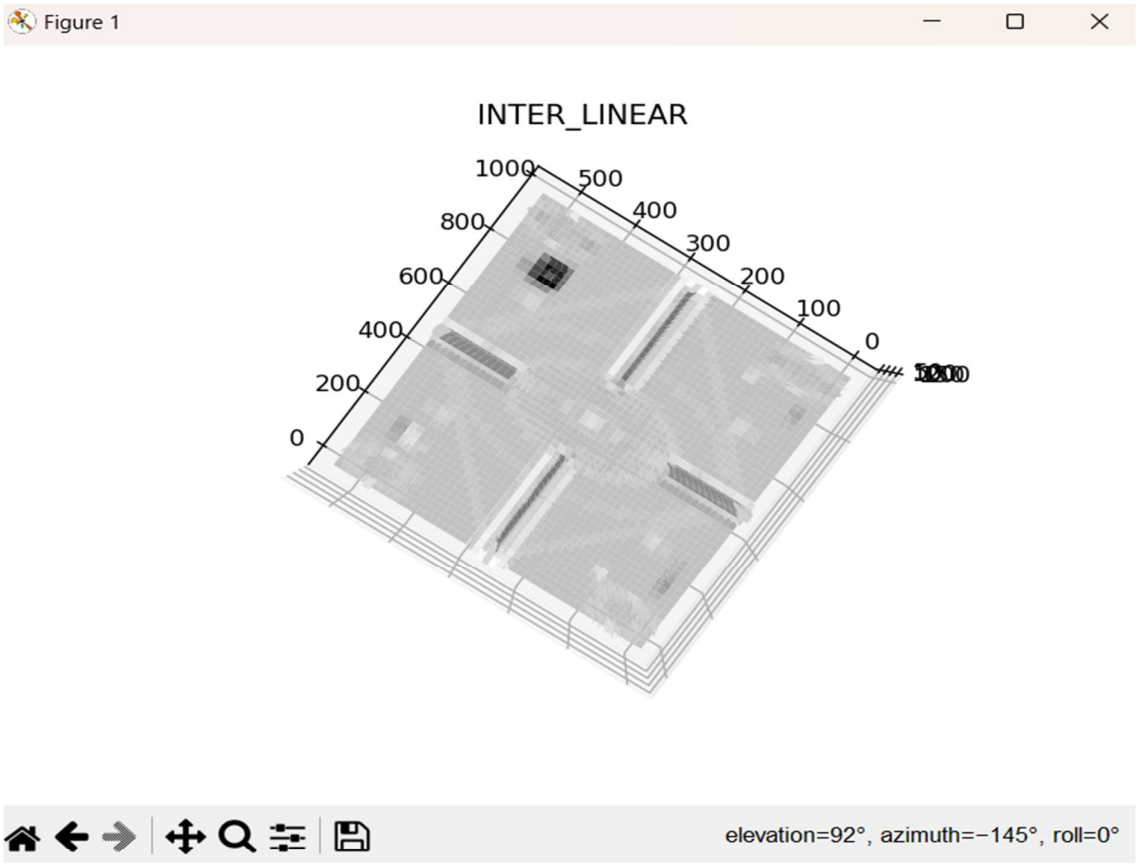


Figure 1
INTER_LINEAR
1000
800
600
400
200
0
500
400
300
200
100
0
elevation=92°, azimuth=−145°, roll=0°
INTER_LINEAR
0
20
40
60
80
100
120
140
200
150
100
50
0
0
50
100
150
200
250

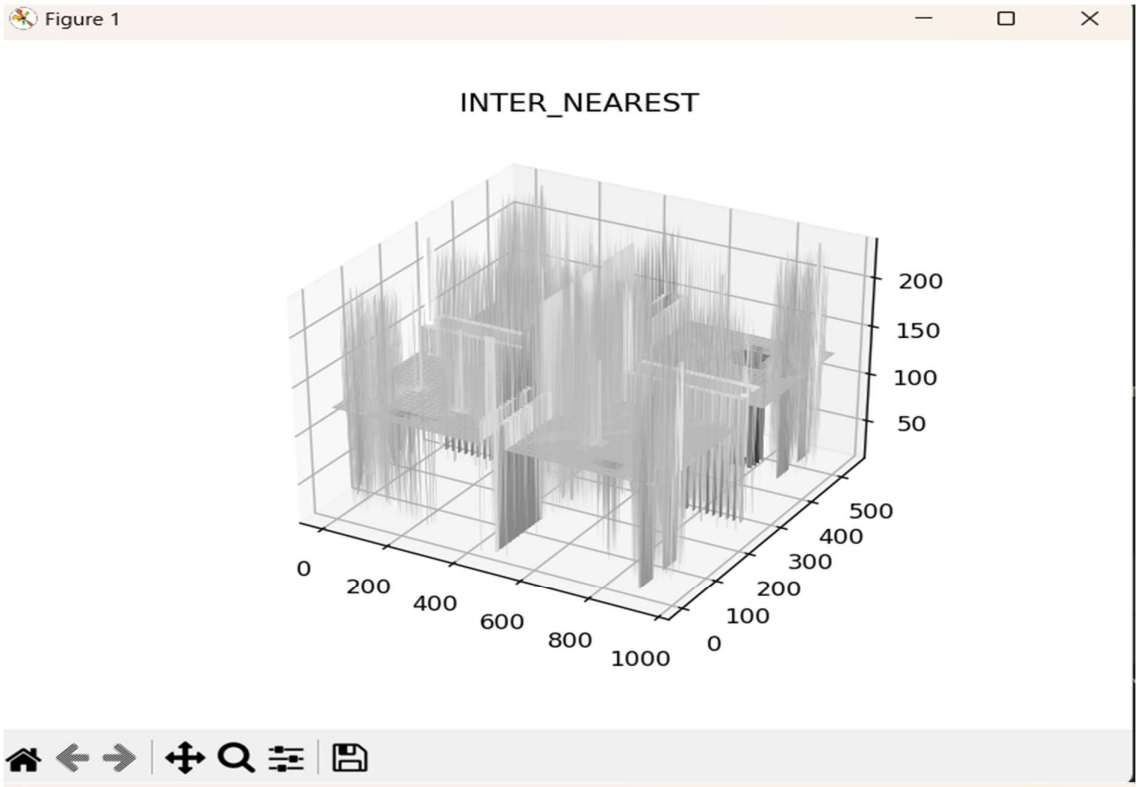


Figure 1
INTER_NEAREST
0
200
400
600
800
1000
0
100
200
300
400
500
50
100
150
200

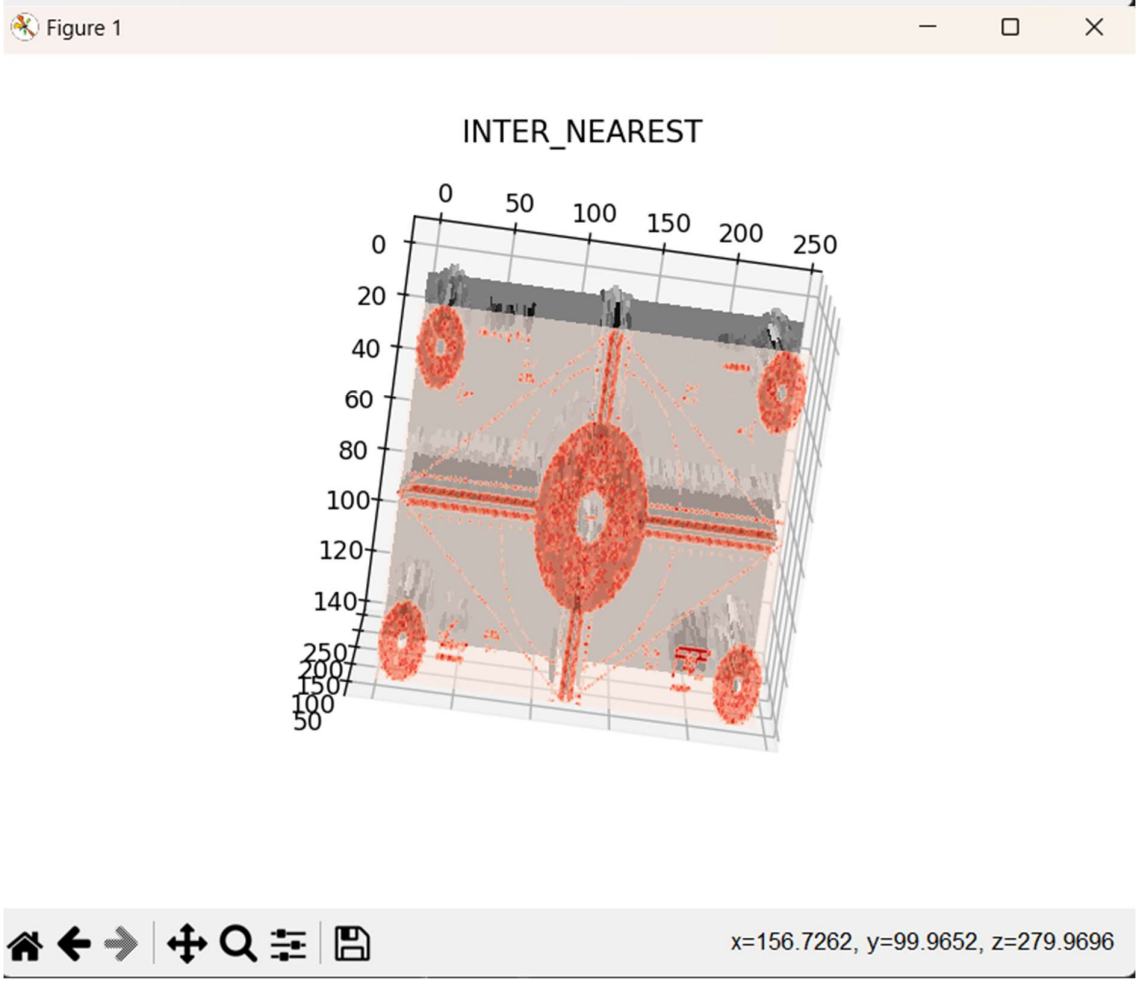


Figure 1
INTER_NEAREST
0
50
100
150
200
250
0
20
40
60
80
100
120
140
250
200
150
100
50
x=156.7262, y=99.9652, z=279.9696

Code :

```
import cv2
import numpy as np
import time
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from concurrent.futures import ThreadPoolExecutor, as_completed


def load_image(image_path):
    img = cv2.imread(image_path)
    if img is None:
        raise FileNotFoundError(f"Failed to load image at {image_path}")
    return img


def resize_with_interpolation(img, output_size):
    with ThreadPoolExecutor() as executor:
        futures = {
            executor.submit(cv2.resize, img, output_size,
interpolation=cv2.INTER_NEAREST): 'INTER_NEAREST',
            executor.submit(cv2.resize, img, output_size,
interpolation=cv2.INTER_LINEAR): 'INTER_LINEAR',
            executor.submit(cv2.resize, img, output_size,
interpolation=cv2.INTER_CUBIC): 'INTER_CUBIC',
        }

        results = {}
        for future in as_completed(futures):
            method = futures[future]
            results[method] = future.result()

    return results['INTER_NEAREST'], results['INTER_LINEAR'],
results['INTER_CUBIC']


def measure_timing(img, output_size):
    timings = {}
    def measure_single_method(interpolation_flag):
        start_time = time.time()
        for _ in range(1000):
            _ = cv2.resize(img, output_size,
interpolation=interpolation_flag)
        end_time = time.time()
        return (end_time - start_time) * 1000  # convert to milliseconds
    for interpolation_flag in [cv2.INTER_NEAREST, cv2.INTER_LINEAR,
cv2.INTER_CUBIC]:
        timings[interpolation_flag] =
measure_single_method(interpolation_flag)
    return timings


def custom_resize(img, output_size):
    img_height, img_width = img.shape[:2]
    new_height, new_width = output_size
    new_img = np.zeros((new_height, new_width, 3), dtype=np.uint8)
    for y in range(new_height):
```

```
        for x in range(new_width):
            src_y = int(y * img_height / new_height)
            src_x = int(x * img_width / new_width)
            new_img[y, x] = img[src_y, src_x]
    return new_img


def display_resized_images(resized_nearest, resized_linear, resized_cubic,
custom_resized):
    fig, axes = plt.subplots(2, 2, figsize=(10, 8))
    fig.suptitle('Resized Images Using Different Interpolation Methods')
    axes[0, 0].imshow(cv2.cvtColor(resized_nearest, cv2.COLOR_BGR2RGB))
    axes[0, 0].set_title('INTER_NEAREST')
    axes[0, 0].axis('off')
    axes[0, 1].imshow(cv2.cvtColor(resized_linear, cv2.COLOR_BGR2RGB))
    axes[0, 1].set_title('INTER_LINEAR')
    axes[0, 1].axis('off')
    axes[1, 0].imshow(cv2.cvtColor(resized_cubic, cv2.COLOR_BGR2RGB))
    axes[1, 0].set_title('INTER_CUBIC')
    axes[1, 0].axis('off')
    axes[1, 1].imshow(cv2.cvtColor(custom_resized, cv2.COLOR_BGR2RGB))
    axes[1, 1].set_title('Custom Resized')
    axes[1, 1].axis('off')
    plt.show()

# Function to plot 3D image with differences highlighted in red
def plot_3d_image(original_img, resized_img, title, downscale_factor=4):
    resized_original = cv2.resize(original_img, (resized_img.shape[1],
resized_img.shape[0]))
    original_gray = cv2.cvtColor(resized_original, cv2.COLOR_BGR2GRAY)
    resized_gray = cv2.cvtColor(resized_img, cv2.COLOR_BGR2GRAY)
    diff = np.abs(original_gray - resized_gray)

    # Downsample the image and the difference for faster plotting
    original_gray = cv2.resize(original_gray, (original_gray.shape[1] //
downscale_factor, original_gray.shape[0] // downscale_factor))
    resized_gray = cv2.resize(resized_gray, (resized_gray.shape[1] //
downscale_factor, resized_gray.shape[0] // downscale_factor))
    diff = cv2.resize(diff, (diff.shape[1] // downscale_factor,
diff.shape[0] // downscale_factor))

    x = np.arange(resized_gray.shape[1])
    y = np.arange(resized_gray.shape[0])
    x, y = np.meshgrid(x, y)
    z = resized_gray

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(x, y, z, facecolors=plt.cm.gray(z / 255), rstride=1,
cstride=1, antialiased=False, shade=False)
    ax.contourf(x, y, diff, zdir='z', offset=z.min() - 20, cmap='Reds',
alpha=0.6)
    ax.set_title(title)
    plt.show()


if __name__ == "__main__":
    input_image_path = r"D:\bhanu\desktop\Projects_working\veipvev\G178_2 -
1080.BMP"
    try:
```

```
        img = load_image(input_image_path)

        output_size = (int(img.shape[1] / 2), int(img.shape[0] / 2))

        resized_nearest, resized_linear, resized_cubic =
resize_with_interpolation(img, output_size)

        timings = measure_timing(img, output_size)
        print(f"Time taken for 1000 iterations using INTER_NEAREST:
{timings[cv2.INTER_NEAREST]:.2f} ms")
        print(f"Time taken for 1000 iterations using INTER_LINEAR:
{timings[cv2.INTER_LINEAR]:.2f} ms")
        print(f"Time taken for 1000 iterations using INTER_CUBIC:
{timings[cv2.INTER_CUBIC]:.2f} ms")

        custom_resized = custom_resize(img, output_size)

        display_resized_images(resized_nearest, resized_linear,
resized_cubic, custom_resized)

        labels = ['INTER_NEAREST', 'INTER_LINEAR', 'INTER_CUBIC']
        times = [timings[cv2.INTER_NEAREST], timings[cv2.INTER_LINEAR],
timings[cv2.INTER_CUBIC]]
        plt.bar(labels, times, color=['blue', 'green', 'red'])
        plt.xlabel('Interpolation Method')
        plt.ylabel('Time (ms)')
        plt.title('Timing for 1000 Iterations of Different Interpolation
Methods')
        plt.show()
        # Plot 3D images
        plot_3d_image(img, resized_nearest, 'INTER_NEAREST')
        plot_3d_image(img, resized_linear, 'INTER_LINEAR')
        plot_3d_image(img, resized_cubic, 'INTER_CUBIC')
        plot_3d_image(img, custom_resized, 'Custom Resized')
    except FileNotFoundError as e:
        print(f"Error: {e}")
    except Exception as e:
        print(f"Error: {e}")
```