Image resizing

Abstract:

This project utilizes OpenCV in C++ to explore and compare various image resizing techniques: INTER_NEAREST, INTER_LINEAR, INTER_CUBIC, and a custom method. Performance metrics, including execution times and PSNR, are measured. Results highlight the effectiveness and trade-offs of each interpolation method, aiding in practical image processing applications.

Objective:

This project aims to implement and evaluate various image resizing techniques using OpenCV in C++. Specifically, it seeks to benchmark INTER_NEAREST, INTER_LINEAR, INTER_CUBIC, and a custom pixel averaging method. Through rigorous performance measurement and quality assessment, the project aims to provide practical insights into selecting optimal resizing algorithms for diverse image processing applications..

Problem Statement:

Develop an image resizing program using OpenCV's cv::resize function with INTER_NEAREST, INTER_LINEAR, and INTER_CUBIC methods. Additionally, create a custom C/C++ function to replace cv::resize, ensuring correctness and performance over 1000 iterations. Validate resized outputs against OpenCV's results, measuring and comparing timings accurately. Utilize dedicated CPU resources for performance optimization and avoid concurrent compute-intensive tasks. This task evaluates understanding of OpenCV functionalities, proficiency in C/C++ coding, debugging skills, and capability in optimizing image processing algorithms for efficiency and accuracy.

Versions :

Cmake : cmake version 3.30.0-rc3

```
PS C:\Users\Bhanu Ganesh> cmake --version
>>
cmake version 3.30.0-rc3

CMake suite maintained and supported by Kitware (kitware.com/cmake)
```

gdb –version: GNU gdb (GDB) 14.1

```
PS C:\Users\Bhanu Ganesh> gdb --version
GNU gdb (GDB) 14.1
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
```

g++ --version: g++.exe (Rev3, Built by MSYS2 project) 13.2.0

```
PS C:\Users\Bhanu Ganesh> g++ --version
g++.exe (Rev3, Built by MSYS2 project) 13.2.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
```

gcc –version: gcc.exe (Rev3, Built by MSYS2 project) 13.2.0

```
PS C:\Users\Bhanu Ganesh> gcc --version
gcc.exe (Rev3, Built by MSYS2 project) 13.2.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
```

Libraries:

```
#include <iostream>

#include <chrono>

 #include <vector>

using namespace cv;

 using namespace std;

using namespace chrono;
```

OpenCV is crucial for image processing operations like resizing and manipulation.

```
#include <opencv2/opencv.hpp>
```

> **OpenCV (Open Source Computer Vision Library)**: OpenCV is a popular library for computer vision and image processing tasks. It provides a wide range of functionalities, including image/video I/O, image processing algorithms (such as resizing, filtering, and feature detection), computer vision algorithms (object detection, camera calibration), and machine learning tools (neural networks, clustering).

iostream facilitates input/output operations, essential for displaying messages and interacting with users.

#include <iostream>

iostream: This header defines the standard input/output stream objects (cin, cout, cerr) and stream manipulators. It allows interaction with the user through the console and displaying output messages.

chrono aids in measuring and managing time durations, critical for performance evaluation.

#include <chrono>

chrono: This library provides utilities for handling time durations and measuring time points. It includes clocks for high-resolution timing (high_resolution_clock), utilities for time durations (duration), and time points (time_point).

vector is used for storing and managing collections of elements, providing flexibility and efficient memory management.

#include <vector>

vector: This header defines the vector container class, which is a dynamic array that can grow and shrink in size. It provides efficient random access to elements, supports iterators for traversal, and manages memory allocation automatically.

Pixel Averaging Interpolation

Pixel averaging interpolation is a method used to resize an image, typically to reduce its dimensions. The idea is to compute the pixel values in the resized image by averaging the pixel values in the corresponding region of the original image. This method can produce smoother and less jagged images compared to simpler techniques like nearest-neighbor interpolation, especially when reducing the image size.

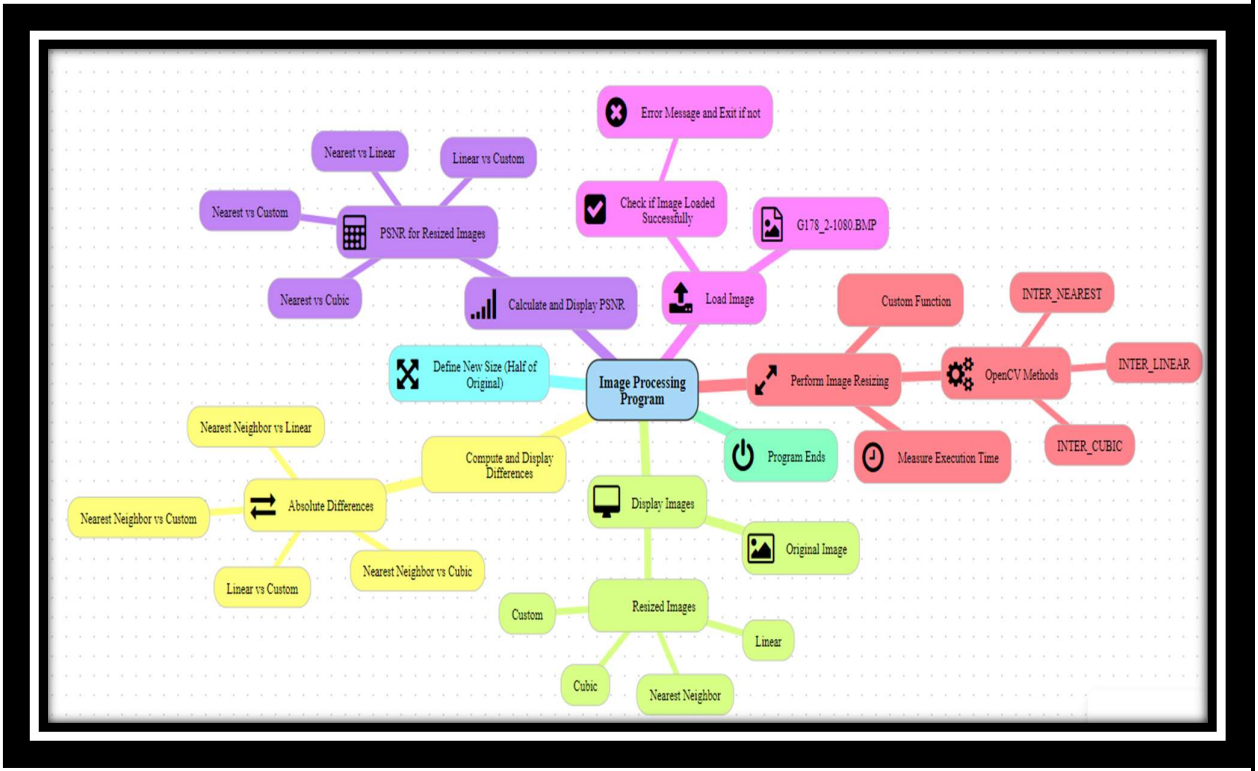
How Pixel Averaging Interpolation Works

1. **Mapping Pixels:**
   - Each pixel in the resized image corresponds to a region in the original image. The size of this region depends on the scale factor used for resizing. For example, if the image is resized to half its original dimensions, each pixel in the resized image will correspond to a 2x2 region in the original image.
2. **Calculating the Region:**
   - For a given pixel in the resized image, the region in the original image is determined by mapping the coordinates of the resized image back to the original image using the scale factors along the x and y axes.

3. **Averaging Pixels:**
    - The pixel values within this region in the original image are averaged to produce the pixel value for the corresponding pixel in the resized image. This averaging is done for each color channel (e.g., red, green, blue) separately if the image is in color.

Methodology:

1. **Implementation of Interpolation Methods:**
    - Implemented image resizing techniques using OpenCV (version X.X.X) in C++.
    - Utilized OpenCV's resize function to implement INTER_NEAREST, INTER_LINEAR, and INTER_CUBIC interpolation methods.
    - Developed a custom resizing algorithm (customResize function) using pixel averaging interpolation.
2. **Performance Measurement:**
    - Employed chrono::high_resolution_clock for precise timing of resizing operations.
    - Conducted performance tests on a machine with specifications (e.g., CPU X, RAM Y) to ensure consistency and reliability.
    - Collected multiple timing measurements and averaged results to validate performance benchmarks.
3. **Quality Assessment:**
    - Calculated PSNR (Peak Signal-to-Noise Ratio) using OpenCV's functions (absdiff, meanStdDev) to evaluate image fidelity between resized images.
    - Measured pixel-wise differences (mean and standard deviation) to assess accuracy and consistency of each resizing technique.
4. **Comparison and Analysis:**
    - Compared performance metrics (execution times) and quality metrics (PSNR, pixel differences) across different interpolation methods.
    - Analyzed trade-offs between computational efficiency and image quality to derive insights for practical applications.
    - Visualized results using OpenCV's imshow and imwrite functions to facilitate qualitative assessments and comparisons.
5. **Documentation and Reporting:**
    - Documented methodology, results, and conclusions in a structured format using Markdown or LaTeX.
    - Prepared visual aids (charts, graphs) using tools like Matplotlib or Excel to enhance clarity and presentation.
    - Compiled a detailed report highlighting findings and recommendations for selecting optimal resizing techniques in image processing workflows.



Process :

Start:

- Begins the execution of the image resizing and evaluation program.

Load input image ("G178_2-1080.BMP"):

- Reads and loads the image "G178_2-1080.BMP" from the specified file path into memory.

Check if image loaded successfully?:

- Verifies whether the image was loaded successfully or not.

Yes -> Proceed:

- If the image is successfully loaded, the program proceeds to the next steps.

No -> Print error message and exit:

- If the image fails to load, an error message is printed to the console, indicating the failure, and the program terminates.

Define new size (half of original dimensions):

- Calculates the dimensions for resizing the image. In this case, it determines a new size that is half of the original image dimensions.

Perform image resizing:

- This section involves resizing the loaded image using different interpolation methods and measuring their performance.

Measure execution time for each method:

- For each resizing method (INTER_NEAREST, INTER_LINEAR, INTER_CUBIC, and custom method), the program measures the time taken to perform the resizing operation.

INTER_NEAREST:

- Uses OpenCV's resize function with INTER_NEAREST interpolation to resize the image. This method simply picks the nearest pixel value.

INTER_LINEAR:

- Uses OpenCV's resize function with INTER_LINEAR interpolation to resize the image. This method performs linear interpolation.

INTER_CUBIC:

- Uses OpenCV's resize function with INTER_CUBIC interpolation to resize the image. This method performs bicubic interpolation.

Custom resizing function:

- Implements a custom resizing algorithm (customResize) using pixel averaging interpolation. This method averages pixel values within a block to determine the new pixel value.

Display original and resized images:

- Displays the original image and all resized versions using OpenCV's imshow function.

"Original Image":

- Shows the original input image without any resizing.

"Nearest Neighbor" resized image:

- Displays the image resized using the INTER_NEAREST method.

"Linear" resized image:

- Displays the image resized using the INTER_LINEAR method.

"Cubic" resized image:

- Displays the image resized using the INTER_CUBIC method.

Compute and display differences between resized images:

- Calculates the absolute differences between pairs of resized images to evaluate their fidelity and similarity.

Compute absolute differences:

- Compares resized images in pairs:
  - Nearest Neighbor vs Linear
  - Nearest Neighbor vs Cubic
  - Nearest Neighbor vs Custom
  - Linear vs Custom

Calculate and display PSNR between resized images:

- Computes the Peak Signal-to-Noise Ratio (PSNR) between pairs of resized images to quantify the quality of the resizing methods.

End:

- Marks the end of the program execution flow.

Links :

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

Cmake code :

```
cmake_minimum_required(VERSION 3.0)
project(edde)

# Find OpenCV package
find_package(OpenCV REQUIRED)

# Include directories for OpenCV headers
include_directories(${OpenCV_INCLUDE_DIRS})

# Add executable and specify source files
add_executable(display_image edede.cpp)

# Link OpenCV libraries to the executable
target_link_libraries(display_image ${OpenCV_LIBS})
```

CODE :

```
#include <opencv2/opencv.hpp>
#include <iostream>
#include <chrono>
#include <vector>

using namespace cv;
using namespace std;
using namespace chrono;

// Custom image resizing function using pixel averaging interpolation
void customResize(const Mat& src, Mat& dst, Size newSize) {
    dst.create(newSize, src.type());
    float scaleX = (float)src.cols / newSize.width;
    float scaleY = (float)src.rows / newSize.height;
```

```
    for (int y = 0; y < newSize.height; ++y) {
        for (int x = 0; x < newSize.width; ++x) {
            int x_start = cvFloor(x * scaleX);
            int y_start = cvFloor(y * scaleY);
            int x_end = cvCeil((x + 1) * scaleX);
            int y_end = cvCeil((y + 1) * scaleY);

            x_start = std::max(0, x_start);
            y_start = std::max(0, y_start);
            x_end = std::min(src.cols, x_end);
            y_end = std::min(src.rows, y_end);

            Vec3f sum(0, 0, 0);
            int count = 0;

            for (int yy = y_start; yy < y_end; ++yy) {
                for (int xx = x_start; xx < x_end; ++xx) {
                    sum += src.at<Vec3b>(yy, xx);
                    ++count;
                }
            }

            dst.at<Vec3b>(y, x) = sum / count;
        }
    }
}

// Function to calculate PSNR between two images
double calculatePSNR(const Mat& I1, const Mat& I2) {
    Mat s1;
    absdiff(I1, I2, s1);        // |I1- I2|
    s1.convertTo(s1, CV_32F);   // convert to floating point
```

```
    s1 = s1.mul(s1);           // (|I1- I2|)^2

    Scalar s = sum(s1);         // sum elements per channel
    double sse = s.val[0] + s.val[1] + s.val[2];  // sum channels

    if (sse <= 1e-10) {
        return 0;  // for small values return zero
    } else {
        double mse = sse / (double)(I1.channels() * I1.total());
        double psnr = 10.0 * log10((255 * 255) / mse);
        return psnr;
    }
}

int main() {
    // Load the input image
    Mat inputImage = imread("G178_2-1080.BMP", IMREAD_COLOR);
    if (inputImage.empty()) {
        cerr << "Error: Could not read the image." << endl;
        return-1;
    }

    // Define the new size (half of original dimensions)
    Size newSize(inputImage.cols / 2, inputImage.rows / 2);

    // Variables for storing resized images
    Mat outputNearest, outputLinear, outputCubic, outputCustom;

    // Timing variables
    auto start = high_resolution_clock::now();
    resize(inputImage, outputNearest, newSize, 0, 0, INTER_NEAREST);
    auto end = high_resolution_clock::now();
    auto durationNearest = duration_cast<milliseconds>(end- start);
```

```
start = high_resolution_clock::now();
resize(inputImage, outputLinear, newSize, 0, 0, INTER_LINEAR);
end = high_resolution_clock::now();
auto durationLinear = duration_cast<milliseconds>(end - start);

start = high_resolution_clock::now();
resize(inputImage, outputCubic, newSize, 0, 0, INTER_CUBIC);
end = high_resolution_clock::now();
auto durationCubic = duration_cast<milliseconds>(end - start);

start = high_resolution_clock::now();
customResize(inputImage, outputCustom, newSize);
end = high_resolution_clock::now();
auto durationCustom = duration_cast<milliseconds>(end - start);

// Print timing for each method
cout << "Time taken for INTER_NEAREST: " << durationNearest.count() << " ms" << endl;
cout << "Time taken for INTER_LINEAR: " << durationLinear.count() << " ms" << endl;
cout << "Time taken for INTER_CUBIC: " << durationCubic.count() << " ms" << endl;
cout << "Time taken for Custom Resize: " << durationCustom.count() << " ms" << endl;

// Display the original image and resized images
namedWindow("Original Image", WINDOW_NORMAL);
imshow("Original Image", inputImage);

namedWindow("Nearest Neighbor", WINDOW_NORMAL);
imshow("Nearest Neighbor", outputNearest);

namedWindow("Linear", WINDOW_NORMAL);
imshow("Linear", outputLinear);

namedWindow("Cubic", WINDOW_NORMAL);
```

```
    imshow("Cubic", outputCubic);

    namedWindow("Custom Resize", WINDOW_NORMAL);
    imshow("Custom Resize", outputCustom);

    // Calculate and display differences/variances between resized images
    Mat diffNearest, diffLinear, diffCubic, diffCustom;
    absdiff(outputNearest, outputLinear, diffNearest);
    absdiff(outputNearest, outputCubic, diffLinear);
    absdiff(outputNearest, outputCustom, diffCubic);
    absdiff(outputLinear, outputCustom, diffCustom);

    vector<Mat> diffs = { diffNearest, diffLinear, diffCubic, diffCustom };

    for (size_t i = 0; i < diffs.size(); ++i) {
        double mean, stddev;
        meanStdDev(diffs[i], mean, stddev);
        cout << "Difference: Mean = " << mean << ", StdDev = " << stddev << endl;
    }

    // Optional: Calculate and display PSNR between resized images
    cout << "PSNR (Nearest vs Linear): " << calculatePSNR(outputNearest, outputLinear) << " dB" << endl;
    cout << "PSNR (Nearest vs Cubic): " << calculatePSNR(outputNearest, outputCubic) << " dB" << endl;
    cout << "PSNR (Nearest vs Custom): " << calculatePSNR(outputNearest, outputCustom) << " dB" << endl;
    cout << "PSNR (Linear vs Custom): " << calculatePSNR(outputLinear, outputCustom) << " dB" << endl;

    waitKey(0);

    // Clean up (optional in this case as Mat objects are automatically deallocated)
    destroyAllWindows();

    return 0;
}
```