Aaron Sprigle

Professor Hong Man

CPE 462

15 May 2025

Image Alteration Project Report

Abstract

The objective of this project was to develop an image processing application using the OpenCV library in C++. The primary goal was to implement a suite of common image processing techniques that could be applied to user-supplied images directly from the terminal. This project serves both as a practical exploration of OpenCV's capabilities and as a demonstration of terminal-based user interactivity in C++ applications. The tools used included OpenCV for image operations, g++ with MSYS2 on Windows for compilation, and Visual Studio Code as the development environment. The application features a terminal-driven menu where the user selects which transformation to apply to a given image. This interactive design allows for more focused testing and usage of individual processing techniques rather than applying all transformations at once.

The project outcomes include the successful application of six distinct image transformations: color inversion, contrast adjustment, histogram equalization, Gaussian blurring, edge detection, and image sharpening. Each transformation is implemented as a standalone function and tested independently. The results of each operation are displayed in a GUI window and saved to disk for analysis. Overall, the project emphasizes modular code design, practical use of external libraries, and command-line application development.

1. Introduction and Objective

Image processing is a crucial field in computer vision, medical imaging, robotics, surveillance, and more. This project aimed to implement a basic yet functional command-line image processor that supports several transformations. The implemented application allows users to load any image from the command line and apply a selected filter based on a menu.

2. Tools and Approach

Programming Language: C++

Libraries: OpenCV 4.1.0, basic C++ libraries

Operating System: Windows 10 using MSYS2 terminal

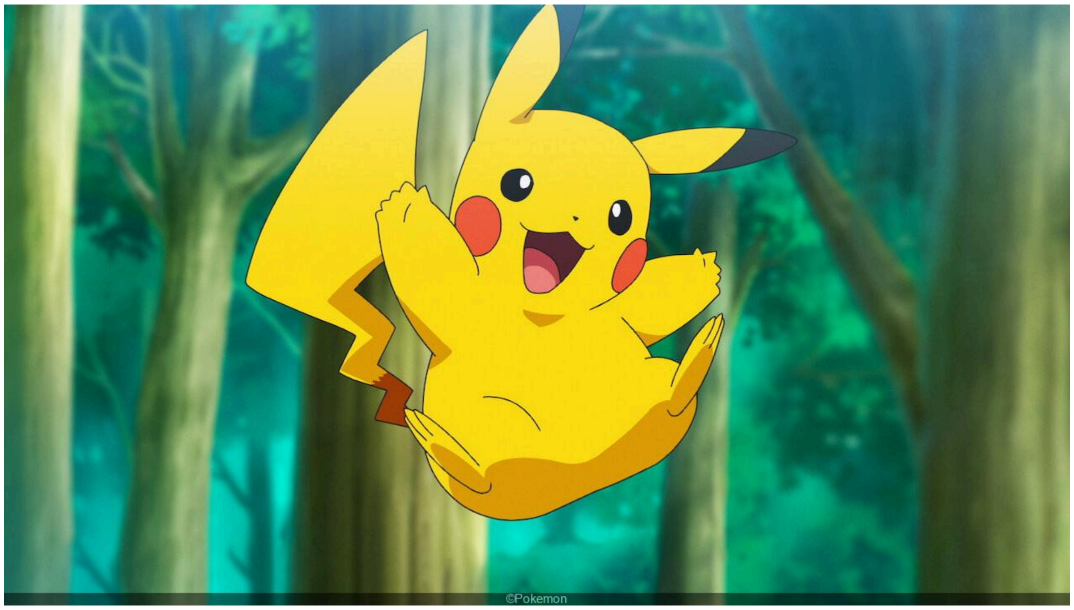
Compiler: g++ (MinGW)

IDE/Editor: Visual Studio Code

The OpenCV library was used for all image processing tasks. After resolving dependency and linking issues with OpenCV, the program was compiled using g++ with the appropriate flags for include and library directories. The final implementation provides a menu-driven interface to apply selected image operations.

3. Features Implemented and Results

Original Image:

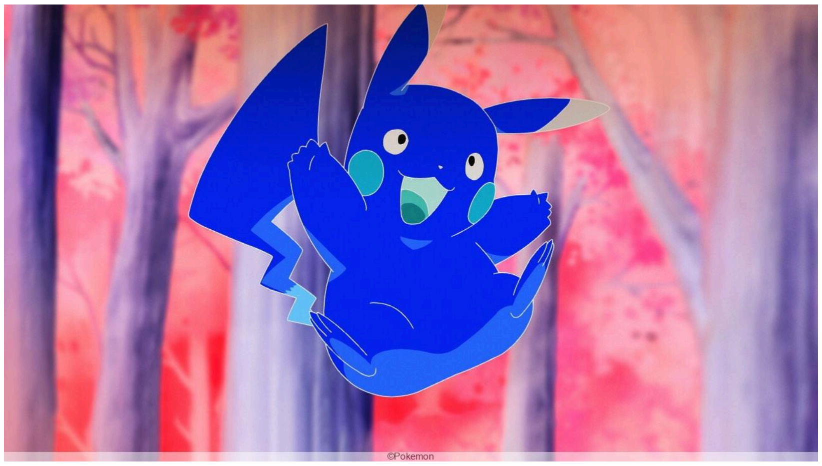


The following image processing features were implemented:

Color Inversion:

Inverts the pixel values of the image.

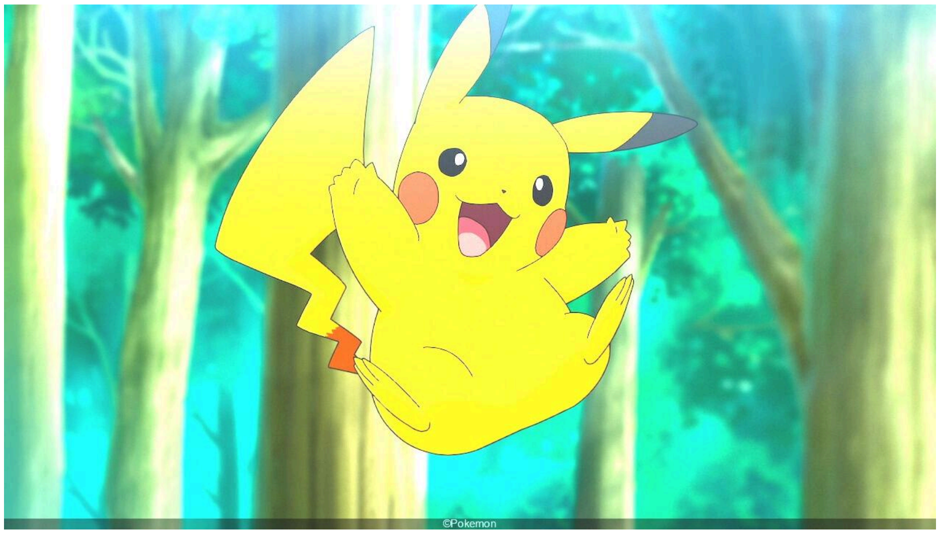
Result image: output_inverted.jpg



Contrast and Brightness Adjustment:

Adjusts contrast with a gain factor (alpha = 1.5) and brightness with a bias (beta = 30).

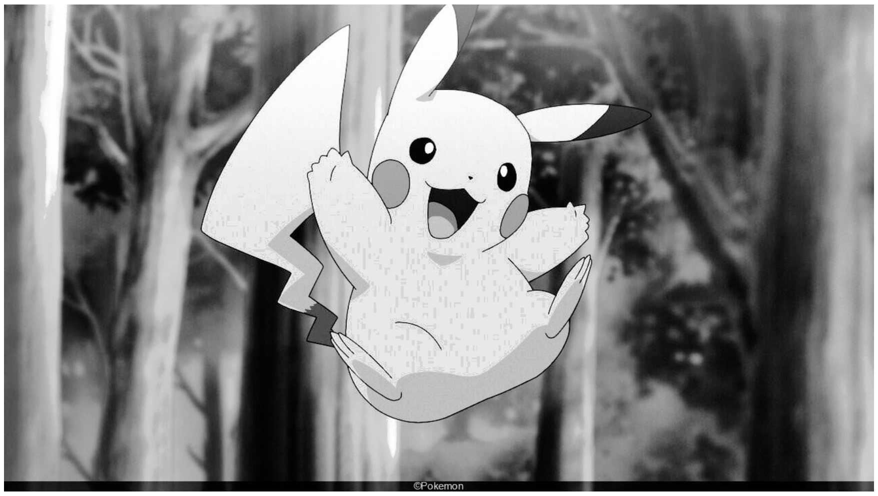
Result image: output_contrast.jpg



Histogram Equalization:

Enhances contrast in grayscale images using OpenCV's equalizeHist.

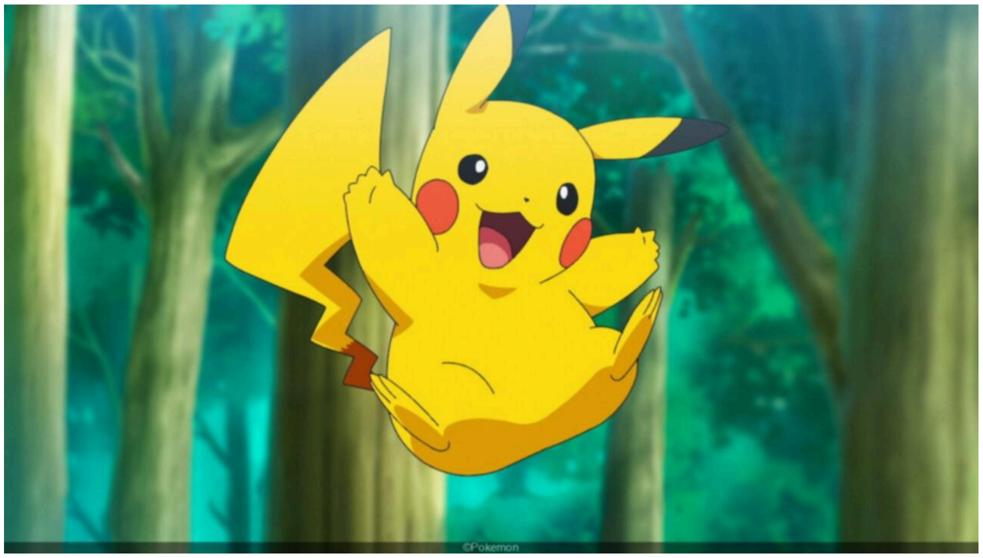
Result image: output_histogram_equalized.jpg



Gaussian Blur (Denoising):

Reduces image noise using a Gaussian kernel of size 5x5.

Result image: output_denoised.jpg



Canny Edge Detection:

Detects edges in grayscale images using OpenCV’s Canny method.

Result image: output_edges.jpg

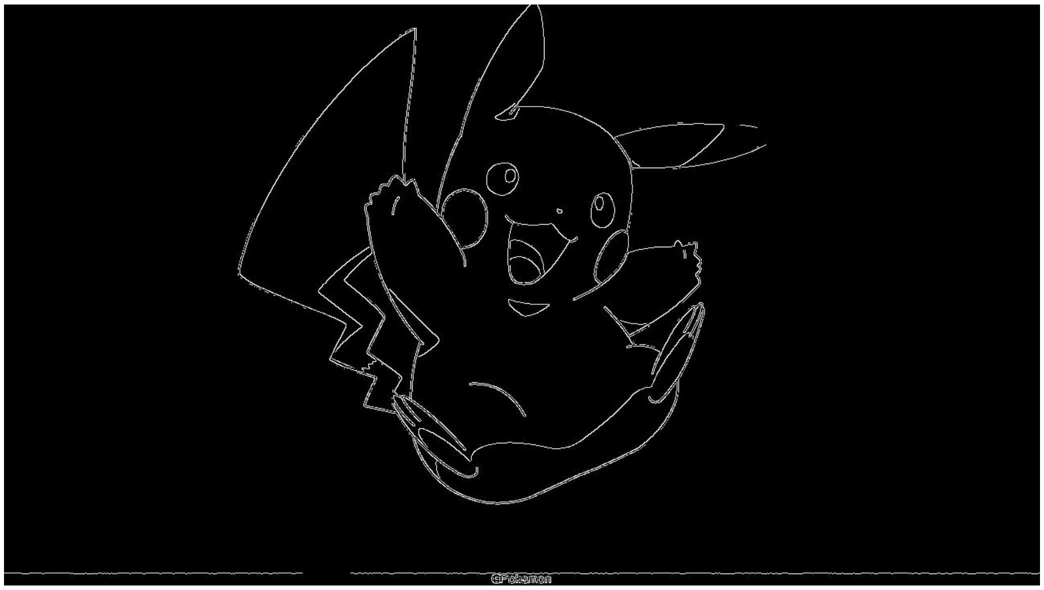


Image Sharpening:

Applies a 3x3 sharpening kernel to enhance edge features.

Result image: output_sharpened.jpg

4. How to Compile and Run

To compile the code on a Windows system using MSYS2 and g++, the following command is used:

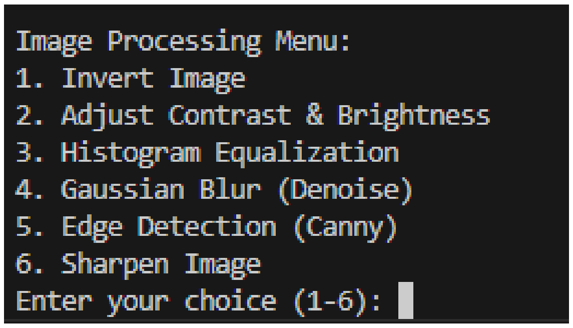
g++ imgpcpj.cpp -I./opencv/build/include -I./opencv/build/include/opencv2 -L./opencv/build/x64/mingw/lib -lopencv_world410 -o imgpcpj.exe

The image of choice for altering can easily be placed in the same directory as the main code “imgpcpj.cpp” to then be edited by the program.

To run the executable:

./imgpcpj.exe <path_to_image>

Once launched, the terminal prompts the user with a menu:



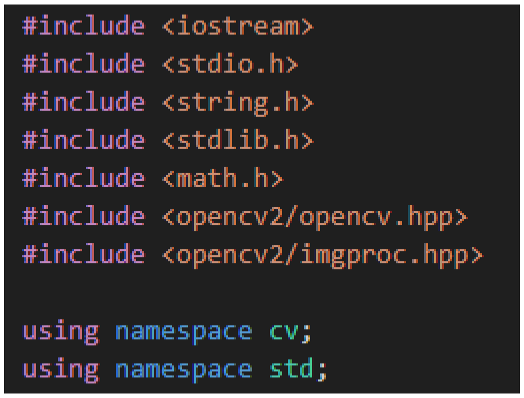
```
Image Processing Menu:
1. Invert Image
2. Adjust Contrast & Brightness
3. Histogram Equalization
4. Gaussian Blur (Denoise)
5. Edge Detection (Canny)
6. Sharpen Image
Enter your choice (1-6):
```

Based on the user's input, the corresponding transformation is applied and displayed using OpenCV's imshow. The result is also saved as a JPEG image in the working directory.

5. Screenshots and Demonstration

Code

Libraries and inclusions:

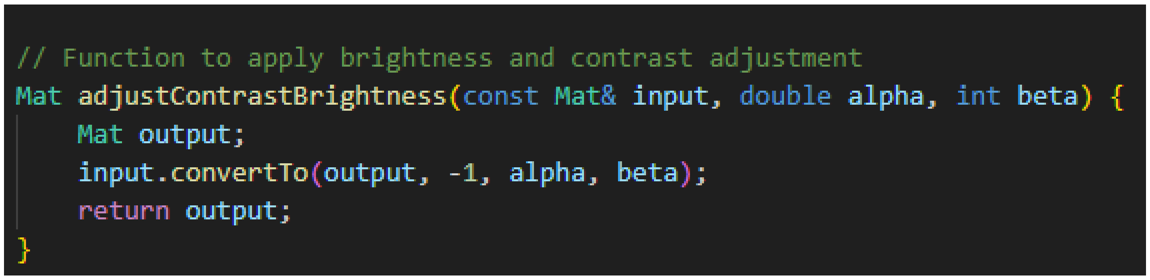


```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <opencv2/opencv.hpp>
#include <opencv2/imgproc.hpp>

using namespace cv;
using namespace std;
```

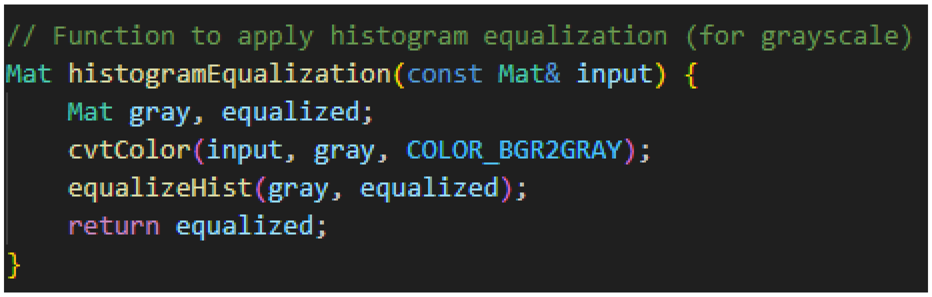
Functions were made for each image transformation except for color inversion

Contrast:



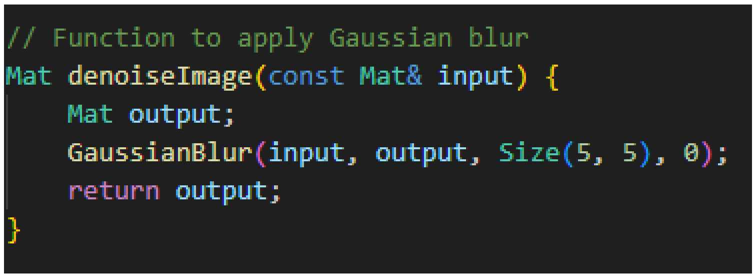
```
// Function to apply brightness and contrast adjustment
Mat adjustContrastBrightness(const Mat& input, double alpha, int beta) {
    Mat output;
    input.convertTo(output, -1, alpha, beta);
    return output;
}
```

Histogram Equalization:



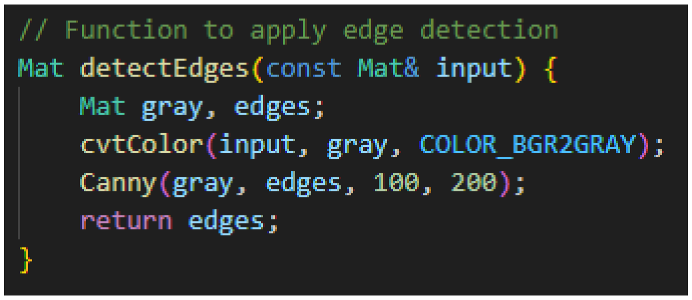
```
// Function to apply histogram equalization (for grayscale)
Mat histogramEqualization(const Mat& input) {
    Mat gray, equalized;
    cvtColor(input, gray, COLOR_BGR2GRAY);
    equalizeHist(gray, equalized);
    return equalized;
}
```

Gaussian Blur:



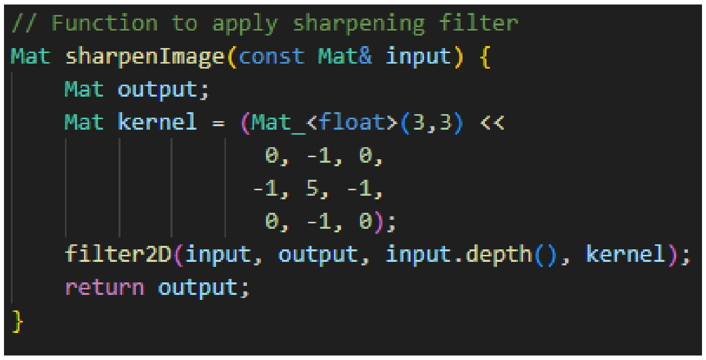
```
// Function to apply Gaussian blur
Mat denoiseImage(const Mat& input) {
    Mat output;
    GaussianBlur(input, output, Size(5, 5), 0);
    return output;
}
```

Edge Detection:



```
// Function to apply edge detection
Mat detectEdges(const Mat& input) {
    Mat gray, edges;
    cvtColor(input, gray, COLOR_BGR2GRAY);
    Canny(gray, edges, 100, 200);
    return edges;
}
```

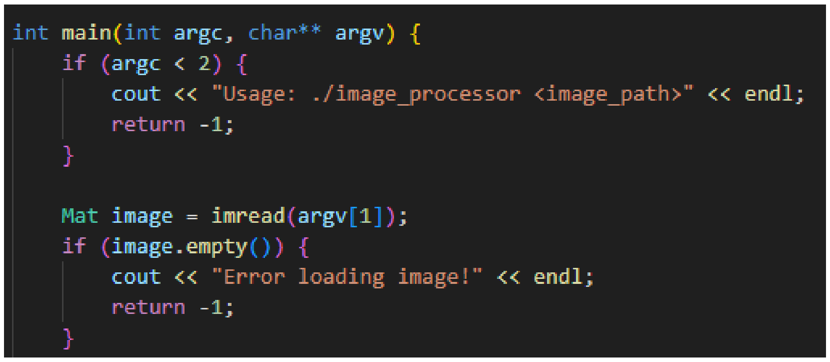
Sharpening:



```cpp
// Function to apply sharpening filter
Mat sharpenImage(const Mat& input) {
    Mat output;
    Mat kernel = (Mat_<float>(3,3) <<
                    0, -1, 0,
                   -1, 5, -1,
                    0, -1, 0);
    filter2D(input, output, input.depth(), kernel);
    return output;
}
```

Main function

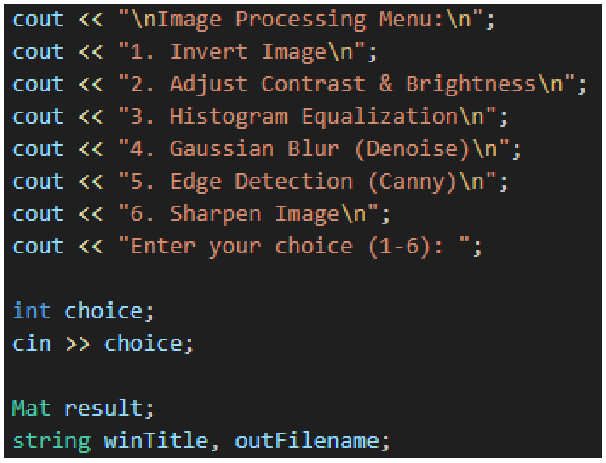
Initializing:



```cpp
int main(int argc, char** argv) {
    if (argc < 2) {
        cout << "Usage: ./image_processor <image_path>" << endl;
        return -1;
    }

    Mat image = imread(argv[1]);
    if (image.empty()) {
        cout << "Error loading image!" << endl;
        return -1;
    }
```

Transformation selection



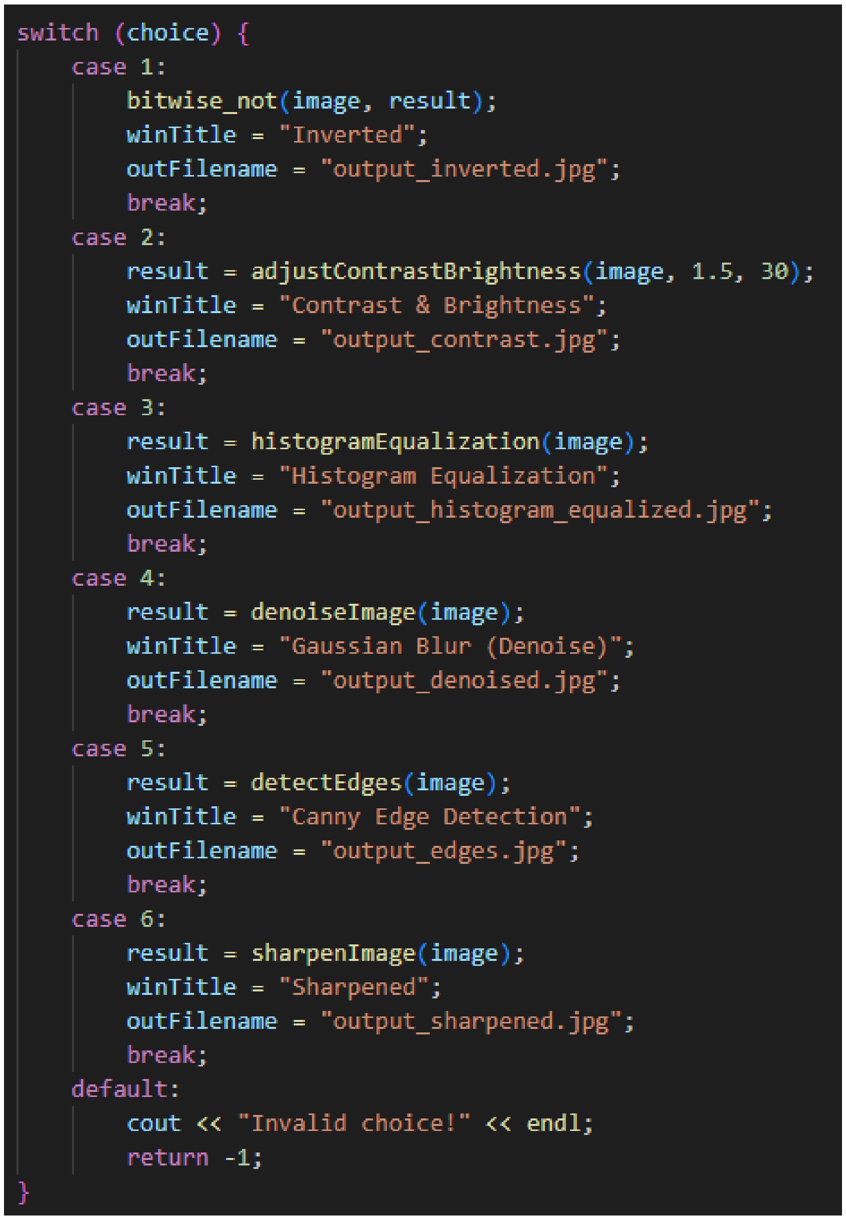
```cpp
cout << "\nImage Processing Menu:\n";
cout << "1. Invert Image\n";
cout << "2. Adjust Contrast & Brightness\n";
cout << "3. Histogram Equalization\n";
cout << "4. Gaussian Blur (Denoise)\n";
cout << "5. Edge Detection (Canny)\n";
cout << "6. Sharpen Image\n";
cout << "Enter your choice (1-6): ";

int choice;
cin >> choice;

Mat result;
string winTitle, outFilename;
```

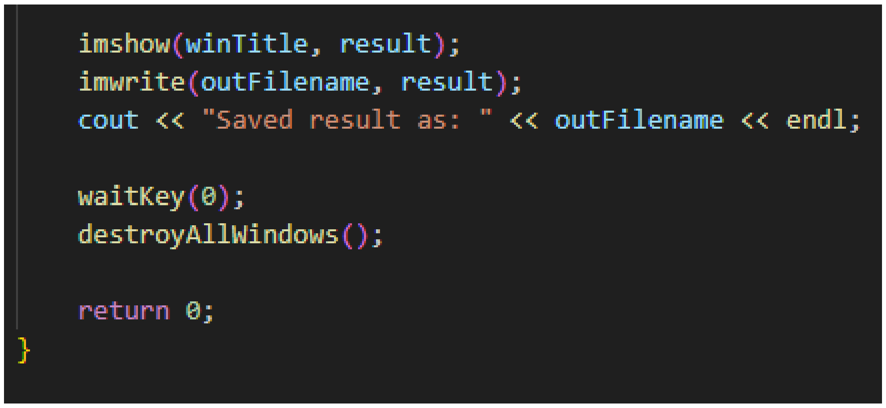
Color inversion is very simple, hence not needing it’s own function

Switch case:



```cpp
switch (choice) {
    case 1:
        bitwise_not(image, result);
        winTitle = "Inverted";
        outFilename = "output_inverted.jpg";
        break;
    case 2:
        result = adjustContrastBrightness(image, 1.5, 30);
        winTitle = "Contrast & Brightness";
        outFilename = "output_contrast.jpg";
        break;
    case 3:
        result = histogramEqualization(image);
        winTitle = "Histogram Equalization";
        outFilename = "output_histogram_equalized.jpg";
        break;
    case 4:
        result = denoiseImage(image);
        winTitle = "Gaussian Blur (Denoise)";
        outFilename = "output_denoised.jpg";
        break;
    case 5:
        result = detectEdges(image);
        winTitle = "Canny Edge Detection";
        outFilename = "output_edges.jpg";
        break;
    case 6:
        result = sharpenImage(image);
        winTitle = "Sharpened";
        outFilename = "output_sharpened.jpg";
        break;
    default:
        cout << "Invalid choice!" << endl;
        return -1;
}
```

Writing new images:



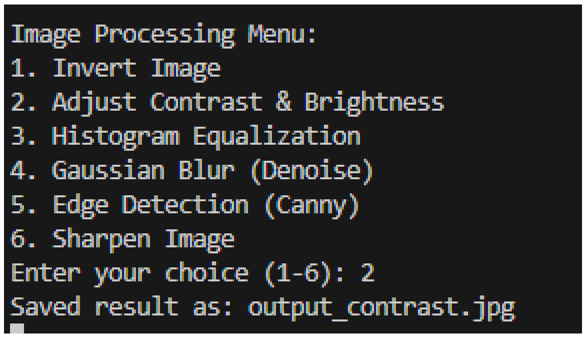
```
    imshow(winTitle, result);
    imwrite(outFilename, result);
    cout << "Saved result as: " << outFilename << endl;

    waitKey(0);
    destroyAllWindows();

    return 0;
}
```

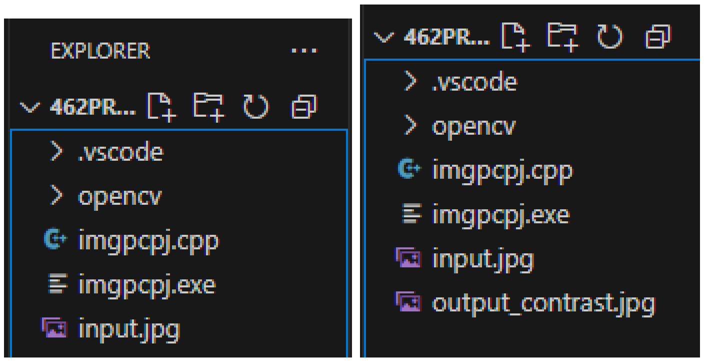
Output

Terminal output after selecting an option:

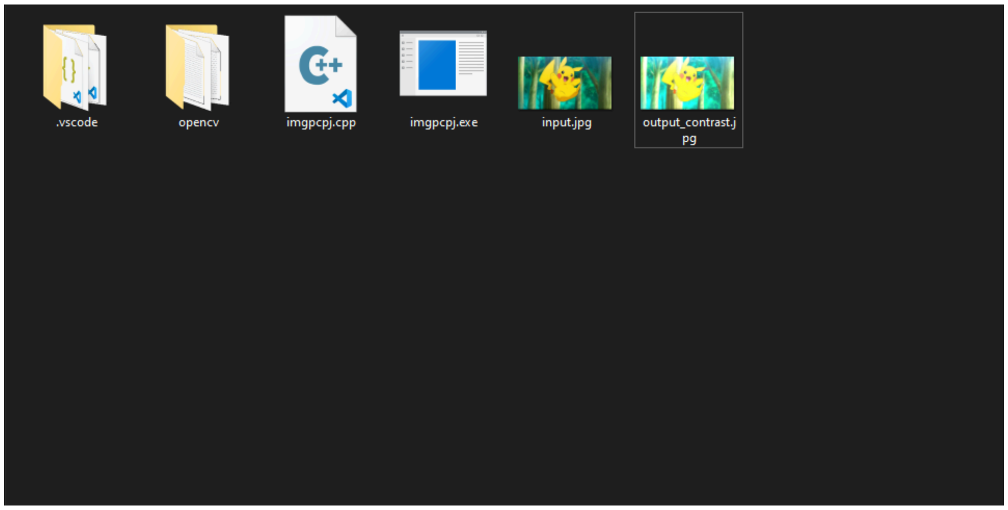


```
Image Processing Menu:
1. Invert Image
2. Adjust Contrast & Brightness
3. Histogram Equalization
4. Gaussian Blur (Denoise)
5. Edge Detection (Canny)
6. Sharpen Image
Enter your choice (1-6): 2
Saved result as: output_contrast.jpg
```

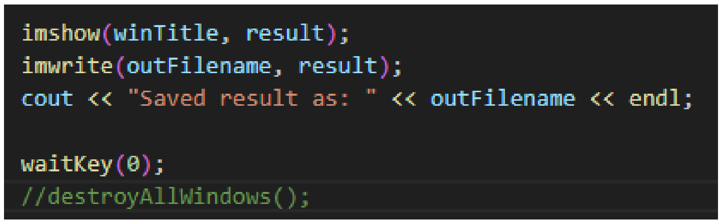
VSCode before and after generating images:



Updated files shown in file explorer

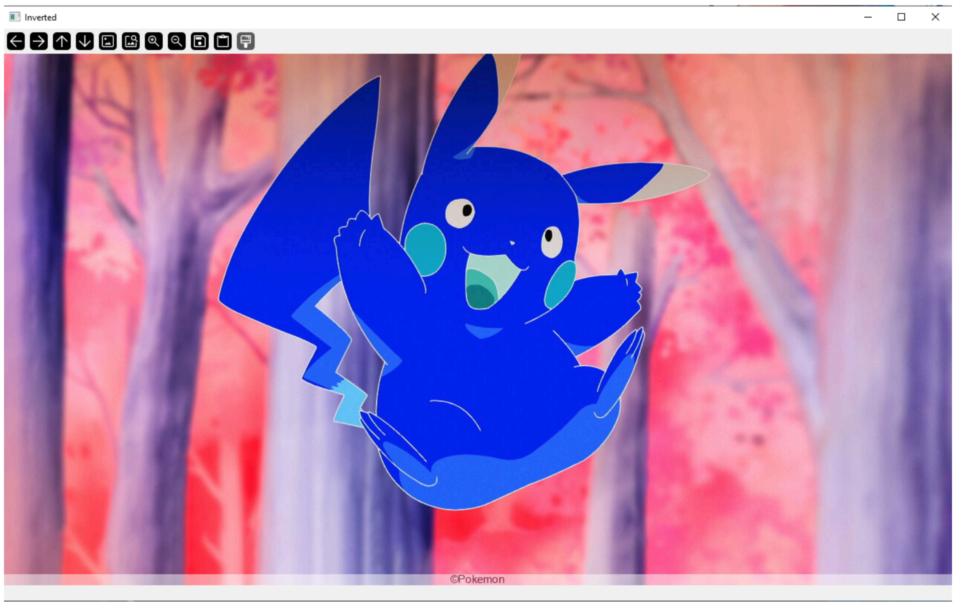


By commenting out or deleting "destroyAllWindows" you can see a preview of the image in a pop up window



```
imshow(winTitle, result);
imwrite(outFilename, result);
cout << "Saved result as: " << outFilename << endl;

waitKey(0);
//destroyAllWindows();
```



6. Contribution

Aaron Sprigle: This was an individual project. Full project implementation, debugging OpenCV linking errors, writing transformation functions, and testing output. All files can be found at my GitHub repository https://github.com/asprigle65/Basic-Image-Processor.

7. Code Attribution

The entirety of the project code was written from scratch. However, the linking and installation of OpenCV followed guidance from official OpenCV documentation and relevant GitHub forums. The core aspects of the project were taken from the code that was implemented in Homework 6 for CPE-462.

8. Conclusion

This project provided valuable experience in using OpenCV with C++ and working with image processing pipelines. There were many issues faced along the way, especially with figuring out OpenCV, but through extensive troubleshooting of dependencies and runtime issues, I was able to better understand image transformations and environment configuration. The final product is functional, interactive, and extensible for additional features such as batch processing or additional filters.