# Basic Photo Editor

Jasmine Owens
*CS5330 Computer Vision*
*The Roux Institute at Northeastern University*
Portland, ME, USA
owens.ja@northeastern.edu

***Abstract*—This project integrates a set of self-built image filters with a basic Graphical User Interface (GUI) in order to provide a primitive photo editing application. In this paper, we will discuss the design of the project, the final implementation, comparable existing products, and areas for future improvement.**

***Index Terms*—image processing, filters, graphical user interface, model-view-controller, exposure, contrast, saturation, color, blue-green-red, hue-saturation-luminosity, GUI, BGR, HLS, MVC**

## I. Introduction

There is an abundance of image editors extant in the world, from the well-known Adobe suite that includes Photoshop [1], to built-in photo editors in Apple [2] or Google [3] Photos, to the plethora of free online tools. Many such tools targeted specifically to editing photos include a standard set of photo editing tools, including exposure, contrast, color saturation, color correction, and others. These tools may also include "effect" filters, such as a grayscale or sepia-tone filter. The impetus for this project was to investigate the process of implementing photo editing software.

As the goal of this project was to create a basic photo editor, there are two main parts to this project:

1) building a set of image filters
2) building a GUI to control the filter effects on an image

As these are separate concerns, the codebases for these two tasks should be separate. As such, GUI functionality is encapsulated in a class independent of the filter class. A main function acts as the interface between the filters and the GUI. This implementation of a model-view-controller (MVC) architecture should allow for easier re-implementation of the GUI at a later date.

This project was implemented in C++ using the OpenCV [4] HighGUI [5] library for the graphical interface. As this libary is not intended to be a full GUI library, the interface is rather limited; as such, the GUI should be viewed as a proof-of-concept rather than a polished product.

Project filters include exposure, contrast, saturation, red tint, green tint, blue tint, grayscale, color inversion, and cartoonize. All of these filters work by directly modifying image values, not by calling OpenCV methods to perform the function.

[1]https://www.photoshop.com
[2]https://www.apple.com/ios/photos/
[3]https://www.google.com/photos/about/
[4]https://opencv.org/
[5]https://docs.opencv.org/3.4/d7/dfc/group__highgui.html

## II. Related Work

A similar project was done by H. Ditrih and S. Grgić in 2020, as documented in the paper "PhotoGrade – fast and effective application for digital photo editing" [1]. Their photo editor was implemented in Java using OpenCV and used the Java Swing framework for the GUI. The editor includes tools to modify image brightness (exposure), contrast, saturation, and tints. It also includes tools to enhance shadows and highlights, as well as a sepia-tone filter. The methods and mathematics behind their implementations do not necessarily match mine.

A 2008 paper by Z. Yang, Y. Zhu and Y. Pu, "Parallel image processing based on CUDA" [2], discusses how general computation on a Graphical Processing Unit (GPU) instead of the Central Processing Unit (CPU) can give a considerable boost in speed for image processing. The paper discusses the technology that makes this possible and demonstrates the speed results for GPU vs CPU computations. Enabling my basic photo editor to use the GPU for computation would likely improve performance; this was not done for the current implementation.

Another possible enhancement to the current implementation would be an option to reduce image noise. Some methods to do this were analyzed in 2016 by S. Tania and R. Rowaida in "A comparative study of various image filtering techniques for removing various noisy pixels in aerial image" [3]. They discuss the pros and cons of the various methods, and show examples of of the filters being applied to an image with various types of noise. This would be a good starting point to implement some sort of noise-reduction filter in a subsequent iteration of this project.

## III. Methods

Image filters and GUI functions are encapsulated in separate classes and integrated in a main function. Together, these three structures form a MVC architecture; the filter set is the Model, the GUI class is the View, and the main loop is the Controller.

The program is launched from command line with the additional argument of path to the image to be modified. The controller loads the image and passes a copy to the GUI. Then, the program enters a continuous loop that applies filters based

on slider values obtained from the GUI and sends the modified image to the GUI in order to update the display.

### A. GUI details

The GUI creates two display windows: a window to display the image, and a window to show the program controls. Upon initialization, the GUI scales the current image to fit to screen. The mechanism by which it does so relies on the ¡windows.h¿ library, and thus is not a portable option. The program terminates when either of the GUI windows is closed.

The control window supports ten sliders and two buttons. Note that OpenCV HighGUI does not implement buttons; these were created using a drawing canvas and mouse events, as described in a post on StackOverflow [4]. The user functions of the sliders and buttons are as follows:

*1) Slider 0:* adjusts the brightness/darkness of the image

*2) Slider 1:* adjusts the threshold for the contrast filter; pixel values in the image will be moved toward or away from the threshold value.

*3) Slider 2:* adjusts the intensity of the contrast filter

*4) Slider 3:* adjusts the color saturation of the image

*5) Slider 4:* adjusts the blue tint of the image

*6) Slider 5:* adjusts the green tint of the image

*7) Slider 6:* adjusts the red tint of the image

*8) Slider 7:* toggles greyscale on and off

*9) Slider 8:* toggles color inversion on and off

*10) Slider 9:* applies cartoonization to the image to the specified level

*11) Button 1:* saves a copy of the current image

*12) Button 2:* resets all sliders to their default positions

No image processing occurs in the GUI except for an image resize when the display image is updated. The "save" function is also handled by the Controller via synchronization of a variable in main() with a GUI variable.

### B. Filter details

The following filters are supported by the GUI: exposure, contrast, saturation, red/green/blue tint, grayscale, color inversion, and cartoonize. The GUI does not directly call the filters; rather, the Controller checks values on the GUI to decide which filters to call.

*1) Exposure:* This filter is implemented as a simple across-the-board increase or decrease in pixel values per channel. The extreme ends of the filter result in a uniformly dark or bright screen.

*2) Contrast:* This filter relies on two GUI sliders: Slider 1 (Threshold) and Slider 2 (Contrast). Pixel values are uniformly incremented towards or away from the Threshold value by the specified amount. The extreme ends of the filter with the default Threshold result either in uniform grey or min/maxed pixel values (i.e., values are either 0 or 255). Results vary for other values of Threshold.

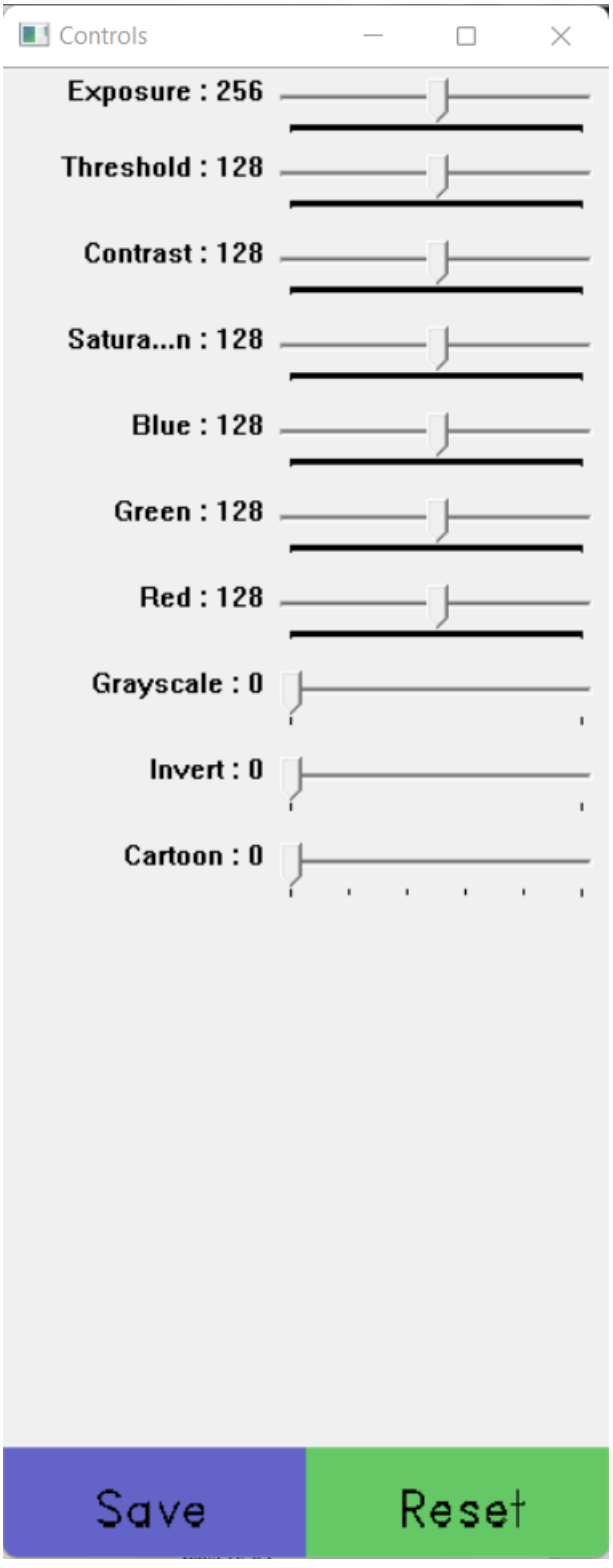


Fig. 1. GUI control window

*3) Saturation:* This filter first converts the image color space from Blue-Green-Red (BGR) to Hue-Luminosity-Saturation (HLS), using the OpenCV color conversion function cvtColor() [6]. In OpenCV, the HLS values are all scaled from 0-255, so the slider value does not need to be scaled, just adjusted to support increase and decrease. The modified slider value is added to the saturation value, and then the image color space is converted back to BGR. The minimum input value results in image grayscale, the maximum results in maximized Sat values across the board.

*4) Tints:* This filter takes in a channel number (1-3) and increases or decreases the pixel values in the specified channel by an adjusted slider value for that channel. The extreme ends of the filter result in a strong Blue/Yellow tint, a strong Green/Magenta tint, or a strong Red/Cyan tint. The filters can also be combined to intensify the tint. If all three tint filters are applied to the same degree, the effect is the same as the Exposure filter.

*5) Grayscale:* I built this filter for a previous project; it simply copies the green-channel values across all three image channels.

*6) Color Inversion:* I built this filter for a previous project; it subtracts the pixel value from 255 in each channel to obtain the inverted image.

*7) Cartoonize:* This filter was also built for a previous project. This filter runs the SobelX and SobelY filters (built for the same project) on the image to detect edges in the image. Then, the filter calculates the Gradient Magnitude (handled in a separate method, built for the same project). Next, color quantization is applied to the image (also a filter built for a previous project). Finally, the gradient magnitude image is thresholded to identify the edges, which are set to black in the color quantized image.

## IV. Results

Figure 2 shows a test image used to illustrate the effects of the filters. I applied a stronger filter effect than might be considered aesthetically pleasing for demonstration purposes. Subtler effects are possible.

Many of the filters implemented for this project operate solely on BGR channels, and as such do not produce particularly realistic results. In particular, the Exposure and Contrast filters tend to skew the color balance as well as the overall darkness and lightness. The contrast filter can produce better results when the threshold is adjusted.

The saturation filter operates on the HLS color space, but still does not produce particularly realistic results. The desaturation operation is more successful than the saturation operation.

The tint filters were all quite successful. Combining tint filters can magnify tint intensities (if the slider values are the same), or if all three are used, can imitate the function of the Exposure filter.

[6]see https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html for color conversion details

Fig. 2. Original image (Nellieville, ME)

Fig. 3. Contrast filter with default threshold

Fig. 4. Contrast filter with increased threshold

Fig. 5. Exposure filter with increase and decrease

Fig. 6. Saturation filter with increase and decrease

Fig. 7. Blue/Yellow tint filter

Fig. 8. Green/Magenta tint filter

Fig. 9. Red/Cyan tint filter

The effect filters were a simpler concept, and this was reflected in the results; the images produced by Grayscale, Color Inversion, and Cartoonize filters were comparatively nicer than those produced by the other filters.

Fig. 10. Effect filters Grayscale and Color Inversion

## V. Discussion

### A. Project highlights

I am most pleased with the tint filters and the effect filters, though the Cartoonize filter is somewhat slow on large images such as Fig. 2. I am also pleased with the button functionality and the successful separation of image processing concerns from the GUI. The separation required some arguably sub-optimal tricks, such as making certain variables used by the GUI controls static. This is fine for this application, since the program can only load one image at a time, but would need to be adjusted should the project be updated to allow opening of multiple files. Such an update may require integrating a full GUI library instead of building up from OpenCV's HighGUI library.

### B. Areas for improvements

An area that I may revisit in the future is the implementation of the Exposure, Contrast, and Saturation filters. In particular, the Exposure filter may give better results if it is implemented on an HLS or similar color space instead of RGB–this would mitigate some of the color alteration issues present in the extant implementation. The Contrast filter may also benefit


Fig. 11. Effect filter Cartoonize

from such a re-implementation. My implementation of the Saturation filter used the same transformation that Ditrih and Grgić [1] used in their project, but operates in the HLS color space instead of the Hue-Saturation-Value (HSV) color space they chose. I did not experiment to see what effect this may have on the filter performance, so this may also be an area to investigate in the future. I did attempt to write my own BGR to HLS conversion methods, but the results were not as good. I would also like to re-build the GUI using a full GUI library in order to have a more polished product. The control window I have is acceptable for demonstrating the effects of my filters, but lacks certain functionality that would be necessary to consider it a finished product. The final improvement I would like to make is in the area of performance. The application slows noticeably when multiple filters are in effect, especially with large image files. The worst offender on its own is the Cartoonize filter, which is not surprising since this filter calls four other filters to produce its output. Improvements may include running the application on the GPU, when available, implementing threading, and C code optimization.

### C. Summary

This project produced a proof-of-concept photo editor with a minimal GUI and primitive filter options. Significant work would need to be done in order to make the software sufficiently robust, elegant, and convenient to be remotely viable. However, the architecture of the project is such that any particular improvement—filter design, GUI design, performance enhancement, etc.—can be implemented without affecting the rest of the code base. In this regard, I would consider this project a success.

## References

[1] H. Ditrih and S. Grgić, "PhotoGrade – fast and effective application for digital photo editing," 2020 International Symposium ELMAR, 2020, pp. 49-52, doi: 10.1109/ELMAR49956.2020.9219012.

[2] Z. Yang, Y. Zhu and Y. Pu, "Parallel image processing based on CUDA," 2008 International Conference on Computer Science and Software Engineering, 2008, pp. 198-201, doi: 10.1109/CSSE.2008.1448.

[3] S. Tani and R. Rowaida, "A comparative study of various image filtering techniques for removing various noisy pixels in aerial image," International Journal of Signal Processing, Image Processing and Pattern Recognition, vol.9, no.3, 2016, pp. 113-124, doi: 10.14257/ijsip.2016.9.3.10.

[4] Miki, "How to make a simple window with one button using OpenCV HighGui only?", StackOverflow, URL (version: 2015-11-26 at 12:27): https://stackoverflow.com/a/33938726