Assignment 1: Simple Image Editor

Kaitlyn Holbert
Affiliation: Undergraduate at Florida Polytechnic University

Report Assignment1-2026-02, Computer Vision

Department of Computer Science
Florida Polytechnic University, Florida, USA

February 2026

ABSTRACT

The subject of this assignment was OpenCV's basic image editing and general operations. Specifically, the main objective was to create an application that can load, edit the brightness and contrast of an image, and save the changes made to it. This was all accomplished using Python in Visual Studio Code.

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

The goal of this project was to create an application that is capable of loading an image, interactively altering its brightness and contrast with scroll bars, and saving the changes to the image. This was done using the OpenCV library in a Python project. The libraries used in this project were OpenCV, numpy, keyboard, and the os module.

To use this application, the user must run the assignment1cb.py file with the opencv, numpy, and keyboard modules installed using pip or another installation method. Four windows will pop up with the images and histograms displayed in color format. To change the brightness and contrast of the image, toggle the scroll bars in the ”New Image” window. To save the image, press ’ctrl’ and ’enter’, and to exit the application press ’L’. The next time the application is run, the edited image will have replaced the old image. To revert back to the original image, close the application without making any edits.

2 IMAGE AND WINDOW SET UP

The original image, edited image, and histograms for each of these images are displayed in four separate windows, created using the namedWindow() OpenCV function. These windows are named Original Image, New Image, Histogram, and New Histogram, respectively. When the sliders are toggled, the window with the original image stays the same, while the window with the edited image and new histogram are updated.

The image is loaded using the OpenCV imread() function. Edited images are saved as 'out.bmp', and the original image, 'dog.bmp', is unchanged by the program. When the image is read, the program first checks if 'out.bmp' exists in the project folder. If it does exist, this image is read and then deleted as to avoid duplicates in the case that the image is changed again. If the image does not exist, then dog.bmp is read instead.

```
if os.path.isfile('out.bmp'):
    image = cv.imread('out.bmp')
    os.remove('out.bmp')
else:
    image = cv.imread('dog.bmp')

if image is None:
    print('Image does not exist.')
    exit(0)
```

Figure 2.1. Code for loading the image into the program

OpenCV 2025

3 BRIGHTNESS AND CONTRAST EDITING

Contrast and brightness are controlled through the alpha and beta parameters, respectively. In this application, two separate scroll bars allow for the control of contrast and brightness levels. The contrast scroll bar has values from 0 to 100, which are divided by 50 in order to find the alpha value, which is between 0 and 2. 0-1 means low contrast, and 1-2 means high contrast. The brightness scroll bar has values between 0 and 255, which represent the beta values directly.

OpenCV's function convertScaleAbs() was used to update the alpha and beta values of the image. As will be discussed in HISTOGRAM ANALYSIS, the two functions for updating the image contain code that updates the histogram for the new image as well.

```
def contrast_action(value):
    if value == 0:
        alpha = 0
    else:
        alpha = value / 50
    global save_image
    save_image = cv.convertScaleAbs(new_image, alpha=alpha)
    cv.imshow('New Image', save_image)

    new_hist_display = np.zeros((256, 256, 3),  dtype=np.uint8)
    new_hist_display[:] = [255, 255, 255]
    hist(save_image, new_hist_display)      # update histogram
    cv.imshow('New Histogram', new_hist_display)
```

Figure 3.1. Function for contrast scroll bar action

OpenCV 2025

```
def brightness_action(value):
    beta = value
    global save_image
    save_image = cv.convertScaleAbs(new_image, beta=beta)
    cv.imshow('New Image', save_image)

    new_hist_display = np.zeros((256, 256, 3),  dtype=np.uint8)
    new_hist_display[:] = [255, 255, 255]
    hist(save_image, new_hist_display)      # update histogram
    cv.imshow('New Histogram', new_hist_display)
```

Figure 3.2. Function for brightness scroll bar action

4 HISTOGRAM ANALYSIS

The histograms displays for both images include three different histograms for each RGB color channel, since 'dog.bmp' is a color image. The OpenCV function calcHist() is used to calculate the histogram, which normalized by another OpenCV function and then drawn on a NumPy image with the line method.

The histogram for the edited image is updated with every change to the brightness and contrast values. The histogram for the original image is only updated if the changes to the original image are saved.

```
def hist(image, hist_image):
    channels = cv.split(image)
    colors = {'B', 'G', 'R'}
    for channel, color  in zip(channels, colors):
        hist = cv.calcHist([channel], [0], None, [256], [0, 256])
        draw_hist(hist, hist_image, color)
```

Figure 4.1. Function for calculating the histograms.

OpenCV 2025

```
def draw_hist(hist, image, color):
    cv.normalize(hist, hist, alpha=0, beta=255, norm_type=cv.NORM_MINMAX)
    for index, value in enumerate(hist):
        if color == 'R':
            cv.line(image, (index, 256), (index, 256 - int(value)), (0, 0, 255), 1)
        elif color == 'G':
            cv.line(image, (index, 256), (index, 256 - int(value)), (0, 255, 0), 1)
        elif color == 'B':
            cv.line(image, (index, 256), (index, 256 - int(value)), (255, 0, 0), 1)
```

Figure 4.2. Function for drawing the histograms. They are first normalized.

5 SAVING CAPABILITIES

To save the image, the user must press the keys ”ctrl+enter”. When this combination is pressed, the image is saved. When the user opens the program again, this new edited image is displayed in place of the original image. In order to close the program, the user could either press the ’L’ key or close all of the windows individually.

The save functionality of this application was implemented with the ’keyboard’ library, specifically with the method ’addhotkey()’. When the specified hotkey is pressed, the displayed image is written to the file ’out.bmp’ with the imwrite() OpenCV method.

```
def save():
    cv.imwrite(filename='out.bmp', img=save_image)

kb.add_hotkey('ctrl+enter', save)
```

Figure 5.1. Code for saving the image

OpenCV 2025

6 COMPLETE PROGRAM

6.1 RESULTS

The final product of this project has image loading, editing, and saving capabilities. The image and a preview are loaded onto the screen, along with their associated histograms. Two scroll bars allow for the increase and decrease of both contrast and brightness. The image can be saved with the shortcut ”ctrl+enter”. Finally, when the code is reloaded, the image is displayed with the new, saved settings.

```
import cv2 as cv
import numpy as np
import keyboard as kb
import os

# default values
alpha = 1.0
beta = 0

# loading image
# ---altered files are called out.bmp
if os.path.isfile('out.bmp'):
    image = cv.imread('out.bmp')
    os.remove('out.bmp')
else:
    image = cv.imread('dog.bmp')

if image is None:
    print('Image does not exist.')
    exit(0)

# new preview image
new_image = cv.convertScaleAbs(image, alpha=alpha, beta=beta)


# trackbar action implementation
# ---contrast is changed by updating alpha
# ---values from 0 to 100 scaled to be between 0 and 2
def contrast_action(value):
    if value == 0:
        alpha = 0
    else:
        alpha = value / 50
    global save_image
    save_image = cv.convertScaleAbs(new_image, alpha=alpha)
    cv.imshow('New Image', save_image)

    new_hist_display = np.zeros((256, 256, 3),  dtype=np.uint8)
    new_hist_display[:] = [255, 255, 255]
    hist(save_image, new_hist_display)      # update histogram
    cv.imshow('New Histogram', new_hist_display)
```

```
# ---brightness is updated by changing beta value to be between 0 and 255
def brightness_action(value):
    beta = value
    global save_image
    save_image = cv.convertScaleAbs(new_image, beta=beta)
    cv.imshow('New Image', save_image)

    new_hist_display = np.zeros((256, 256, 3),  dtype=np.uint8)
    new_hist_display[:] = [255, 255, 255]
    hist(save_image, new_hist_display)      # update histogram
    cv.imshow('New Histogram', new_hist_display)


# saves image
# ---ctrl+enter saves the image
def save():
    cv.imwrite(filename='out.bmp', img=save_image)


# splits channels and calculates histograms for all of them
def hist(image, hist_image):
    channels = cv.split(image)
    colors = {'B', 'G', 'R'}
    for channel, color  in zip(channels, colors):
        hist = cv.calcHist([channel], [0], None, [256], [0, 256])
        draw_hist(hist, hist_image, color)


# draws a color image histogram
# ---done with line fucntion, for all RGB channels
def draw_hist(hist, image, color):
    cv.normalize(hist, hist, alpha=0, beta=255, norm_type=cv.NORM_MINMAX)
    for index, value in enumerate(hist):
        if color == 'R':
            cv.line(image, (index, 256), (index, 256 - int(value)), (0, 0, 255), 1)
        elif color == 'G':
            cv.line(image, (index, 256), (index, 256 - int(value)), (0, 255, 0), 1)
        elif color == 'B':
            cv.line(image, (index, 256), (index, 256 - int(value)), (255, 0, 0), 1)
```

```
#empty np images for drawing histgram on
hist_display = np.zeros((256, 256, 3), dtype=np.uint8)
new_hist_display =  np.zeros((256, 256, 3), dtype=np.uint8)
hist_display[:] = [255, 255, 255]
new_hist_display[:] = [255, 255, 255]

#call function for displayng the histograms for both images
hist(image, hist_display)
hist(new_image, new_hist_display)

# display four windows for images and histograms
cv.namedWindow('Original Image', cv.WINDOW_AUTOSIZE)
cv.imshow('Original Image', image)

cv.namedWindow('New Image', cv.WINDOW_AUTOSIZE)
cv.imshow('New Image', new_image)

cv.namedWindow('Histogram', cv.WINDOW_AUTOSIZE)
cv.imshow('Histogram', hist_display)

cv.namedWindow('New Histogram', cv.WINDOW_AUTOSIZE)
cv.imshow('New Histogram', new_hist_display)

# trackbars
# ---contrast will be from 0 to 2, with 50 being 1
cv.createTrackbar('Contrast', 'New Image', 0, 100, contrast_action)
cv.createTrackbar('Brightness', 'New Image', 0, 255, brightness_action)

# keyboard shortcut for saving
kb.add_hotkey('ctrl+enter', save)

# program ends when windows are closed
while True:
    key = cv.waitKey(0)
    if key == ord('l'):
        cv.destroyAllWindows()
        break
```

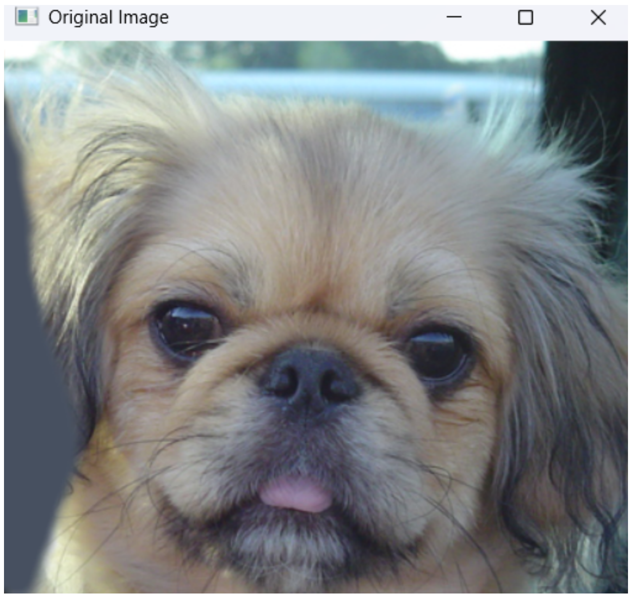


Figure 6.1. Original image, loaded in the ’Original Image’ window

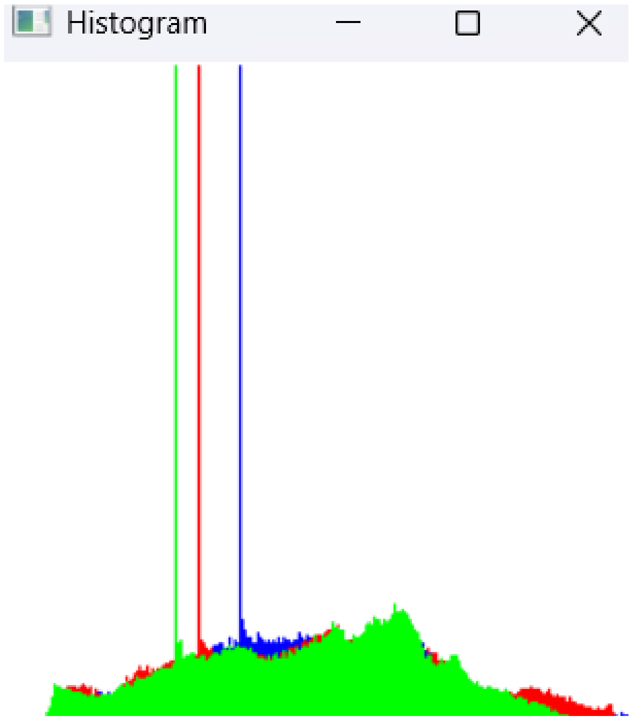


Figure 6.2. Histogram for original image, displayed in ’Histogram’ window

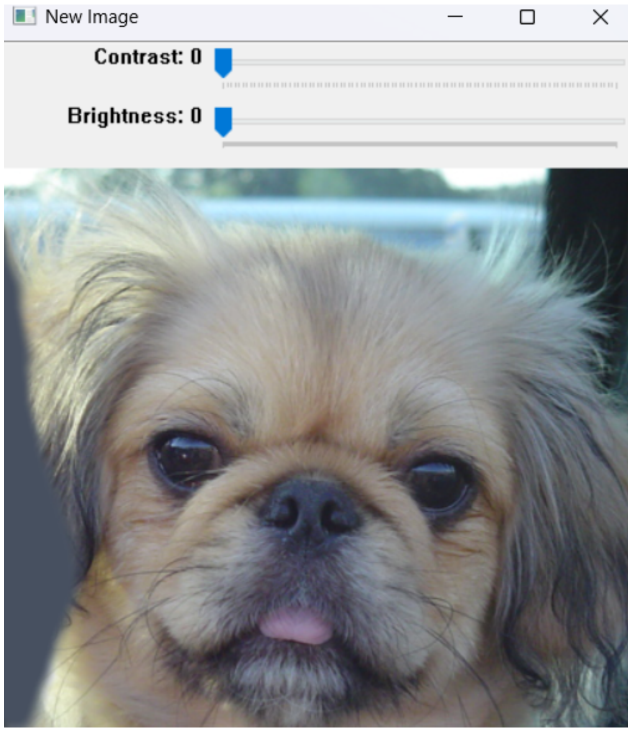


Figure 6.3. The image loaded into the preview window, labeled 'New Image'

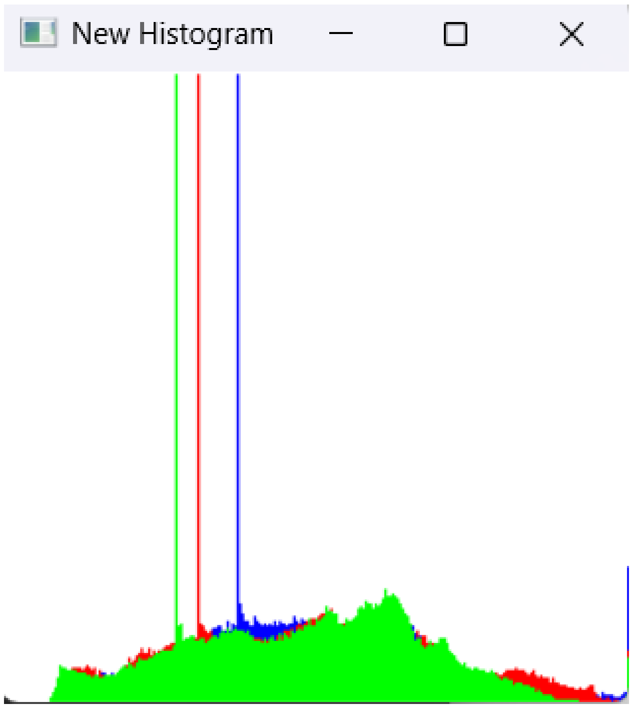


Figure 6.4. The histogram for the new image, since nothing has been done yet, this is the same as the histogram associated with the unedited image.

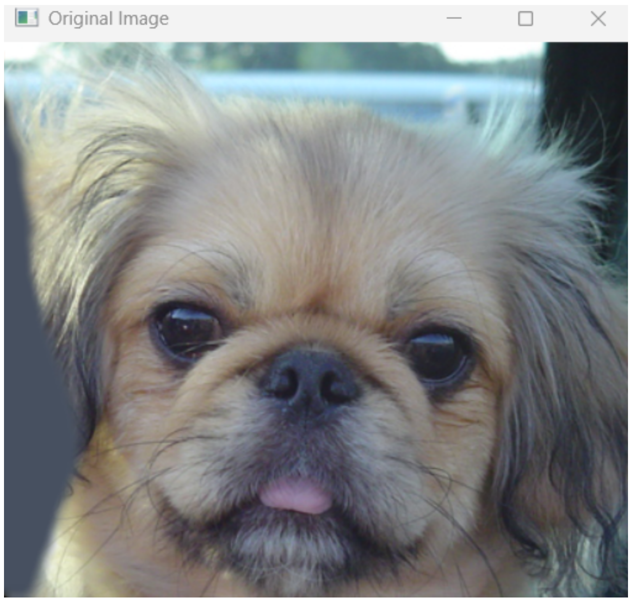


Figure 6.5. The Original Image window after edits have been made in the preview window 'New Image'

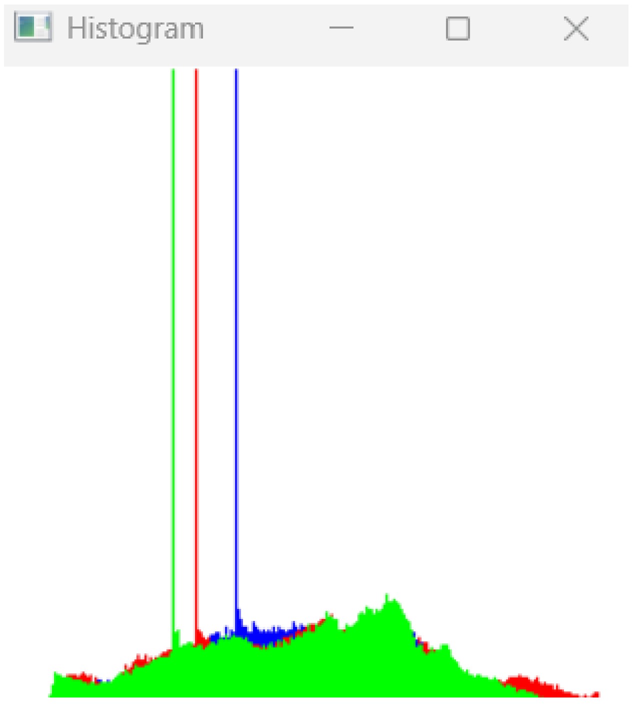
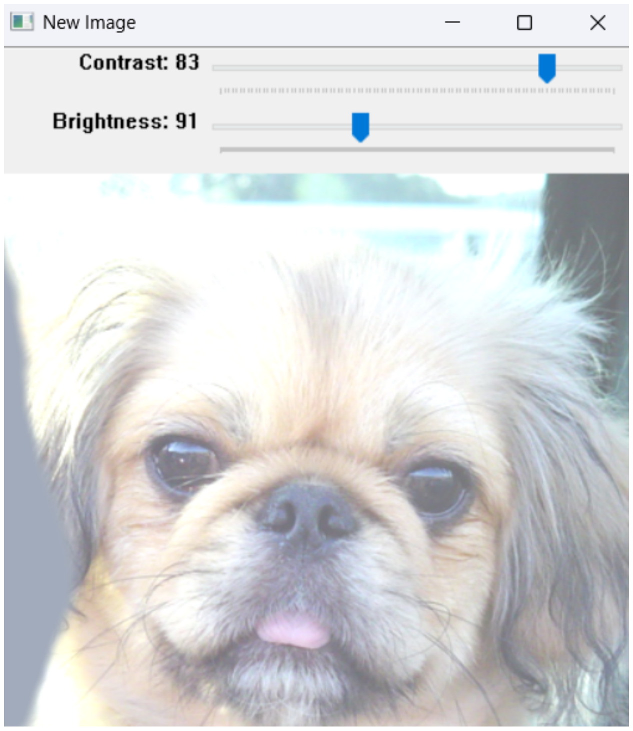


Figure 6.6. The histogram for the original image after edits have been made



Contrast: 83

Brightness: 91

Figure 6.7. The New Image window after edits have been made

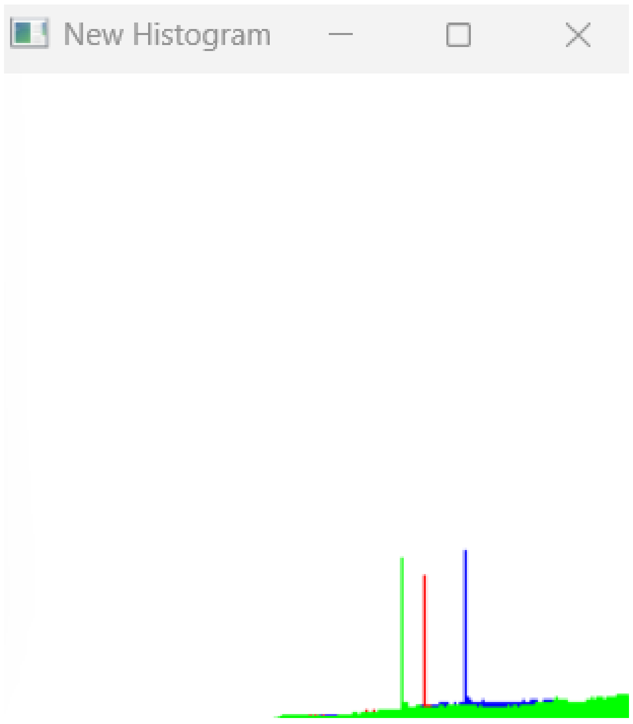


Figure 6.8. The histogram for the new image after edits have been made

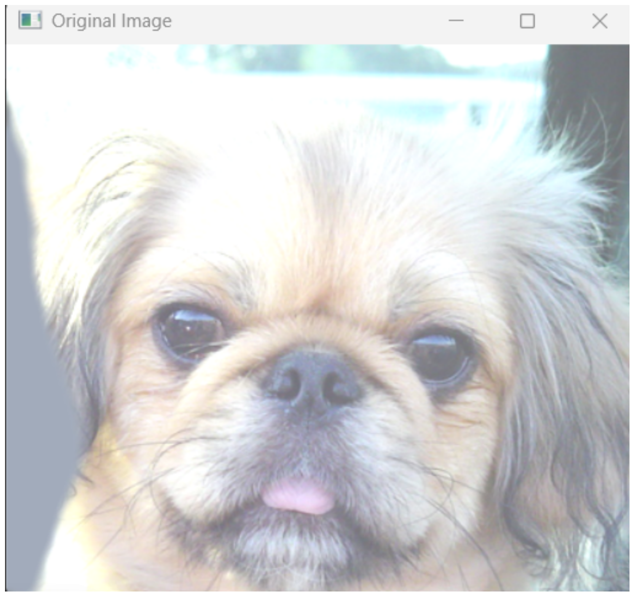


Figure 6.9. The Original Image window after the program has been reloaded

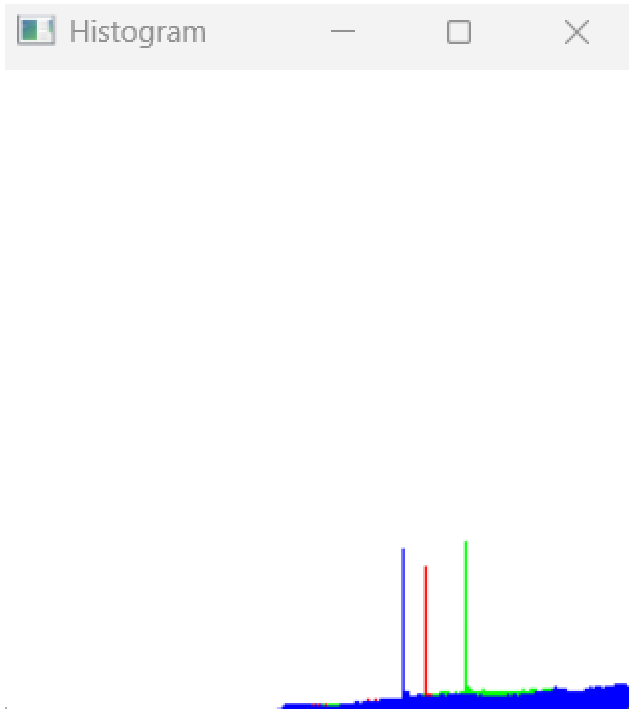


Figure 6.10. The histogram for the original image after the program has been reloaded

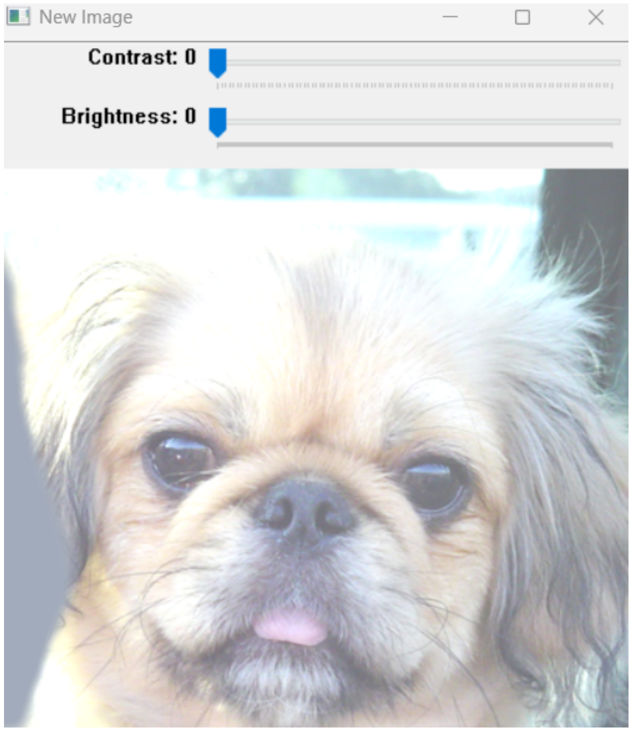


Figure 6.11. The New Image window after the program has been reloaded

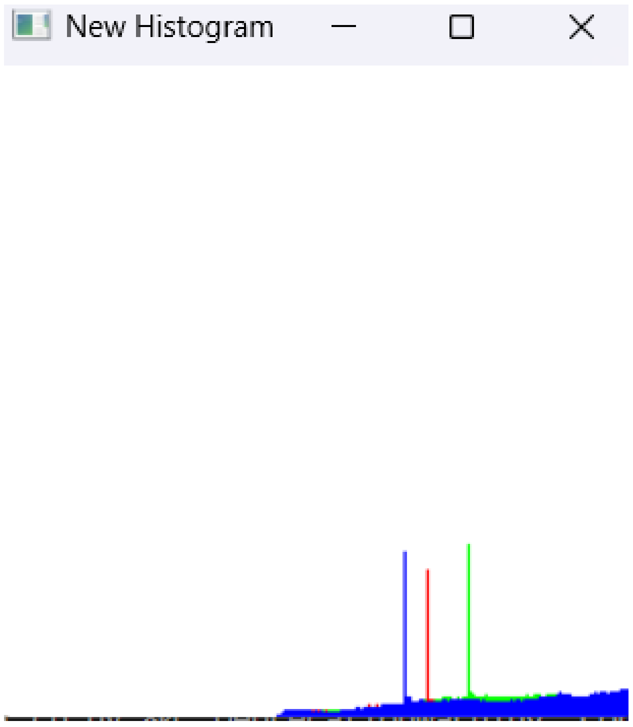


Figure 6.12. The histogram for the new image after the program has been reloaded

REFERENCES

OpenCV (2025). “Opencv modules, <https://docs.opencv.org/3.4/index.html> (June). The OpenCV documentation was used as a reference for the use of all OpenCV methods used in this application.