Assignment 2 Report

Presented

To the Professor of Florida Polytechnic University,

In Fulfillment of the Requirements
for the Computer Vision Brightness and Contrast Image Adjustment

Computer Engineering

Supervised By: Dr. Abid

Written By: Paul Patullo

Date: February 23, 2025

ABSTRACT

This assignment focuses on implementing and applying various image filters to analyze their effects on a given image. The task involves creating custom filters discussed in class including Box, Sobel, and Gaussian filters. There are two types of implementations of the same filter, one being OpenCV's built in function and the other is a manual code. The filters to be implemented are:

Box Filter (without OpenCV's built-in function) for 3x3 and 5x5 image window sizes. Box Filter (with OpenCV's built-in function). Sobel Filter for X-axis edge detection (manual implementation). Sobel Filter for Y-axis edge detection (manual implementation). Sobel Filter for both X and Y-axis edges (manual implementation). Sobel Filter for both X and Y-axis edges (with OpenCV). Gaussian Filter (with OpenCV).

The goal of the assignment is to apply each filter to the image. The focus is on understanding the details, such as sliding window size and convolution process, while comparing custom implementations with OpenCV solutions.

ACKNOWLEDGMENTS

I would like to thank the developers and contributors of the libraries used in this project, including Tkinter, OpenCV, and Matplotlib, for providing powerful tools that allowed us to utilize their libraries in making this assignment. I would also like to acknowledge the open-source community for their resources and documentation, which helped guide me through the coding process.

CONTENTS

1 INTRODUCTION

1.1 THE IMAGE

This project required creating applying multiple filters on two images, a dog and a bike for the purpose of understanding computer vision concepts.

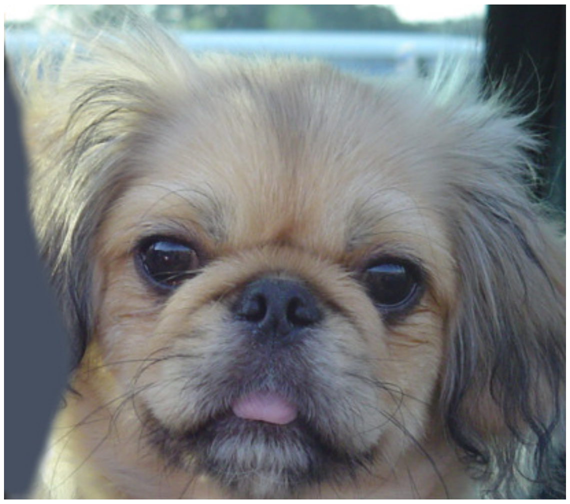

Figure 1.1. A picture of a dog

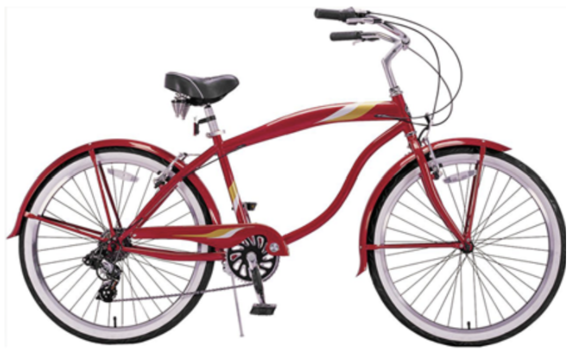

Figure 1.2. A picture of a bike

1.2 FIRST STEPS

When first approaching this assignment, I imported the opencv and tkinter libraries into python for the purpose of creating a GUI to place my original and filtered images. Some functions that came from this consisted of:

```
        tk.Tk()
ImageTk.PhotoImage(image)
```

cv2.imread(pathtoimage.jpg, cv2.IMREADGRAYSCALE)

1.3 PART 0: GUI

When first starting this task, i researched in geeksforgeeks how to create a basic GUI for displaying my images with a filter. I needed to create a window in a grid like pattern to store multiple frames within that i can place my images into and control their specific pixel positions. My basic GUI window looks like this:

Figure 1.3. A picture of blank interface

1.4 PART 1: BOX FILTER

After researching various solutions, I discovered an approach involving the use of the Tkinter library. This library enabled me to create a single window with many distinct frames: one for each image (original and copy/copies) and one for the second image and its filtered image. I noticed early on that I would be important many images into this window with different filters. With this knowledge, I created a function that allowed me to add an image to my window very easily regardless of the filter type. The function parameters covered the image type, grid positioning, image size, and filter size.

Here is the add image() function below:

```
def add_image(image_path, row, col, width=200, height=200, apply_filter=False, k=3):
    if isinstance(image_path, str):
        if apply_filter:
            # Displays filter over image
            filtered_image = boxFilter(image_path, k)
            image = Image.fromarray(filtered_image)
        else:
            # Displays original image
            image = Image.open(image_path)
    else:
        image_matrix = image_path
        image_matrix = cv2.normalize(image_matrix, None, 0, 255, cv2.NORM_MINMAX)
        image_matrix = np.uint8(image_matrix)
        image = Image.fromarray(image_matrix)

    image = image.resize((width, height))
    convert_image = ImageTk.PhotoImage(image)

    label = tk.Label(frame, image=convert_image, bg="white")
    label.image = convert_image

    # Position image using grid
    label.grid(row=row, column=col, padx=30, pady=10)
```

Next is the code for implementing the box filter built without using the openCV command:

```
def boxFilter(image_path, k):  # Box filter algorithm
    image_cv2 = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    copy_matrix = np.copy(image_cv2)
    empty_matrix = np.zeros_like(copy_matrix)
    mask = (1/(k**2)) * np.ones((k, k))

    rows, cols = copy_matrix.shape

    for i in range(rows - k + 1):
        for j in range(cols - k + 1):
            sub_matrix = copy_matrix[i:i+k, j:j+k]
            filter = np.sum(sub_matrix * mask)
            empty_matrix[i + k//2, j + k//2] = filter

    return empty_matrix
```

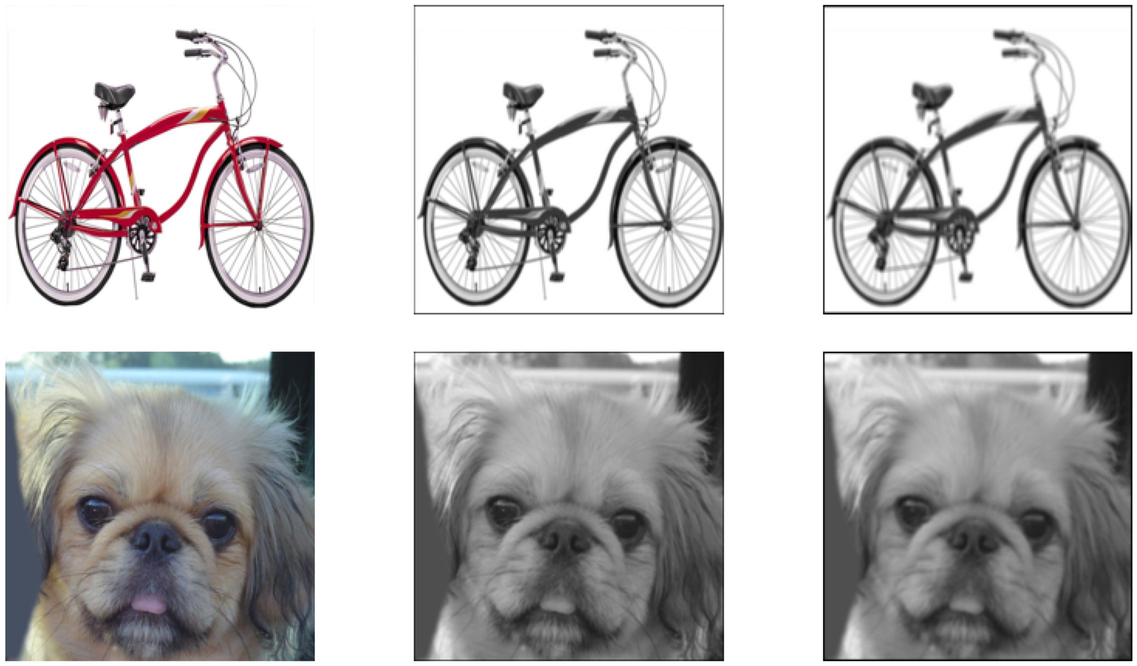

Figure 1.4. Box Filter no OpenCV

1.5 PART 2: BOX FILTER OPENCV

Next I implemented the filter easily with the short command of:

```
def box_filter_command(image_path, k):
    image_cv2 = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    blurred_image = cv2.blur(image_cv2,(k,k))
    return blurred_image
```

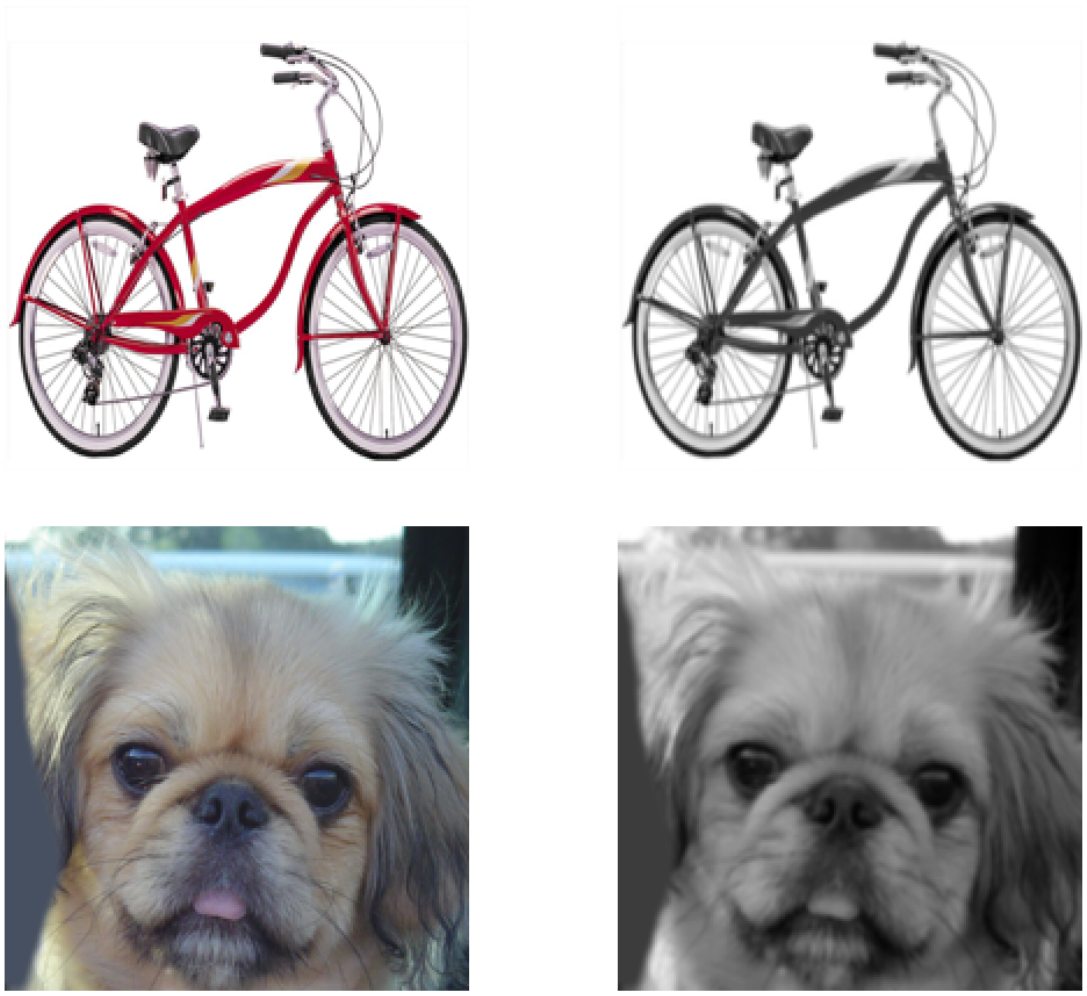

Figure 1.5. Box Filter with OpenCV

1.6 PART 3: SOBEL X AXIS NO OPENCV

This part took some research but once i understood I needed to covnert the image to grayscale. The process became easier, but first I needed t ocreate to plan to implement this. I needed an empty matrix to store the data pulled from the sliding window, and once I had this created, I could create a new image from each pixel filtered. Here is the code for this function:

```
def sobel_filter_x_command(image_path):
    image_cv2 = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    kernelx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    copy_matrix = image_cv2
    empty_matrix = np.zeros_like(copy_matrix)
    width, height = image_cv2.shape

    #algorithm for convoluting each pixel
    for i in range(1, width - 1):
        for j in range(1, height - 1):
            sub_matrix = copy_matrix[i-1:i+2, j-1:j+2] #Extracts a 3x3 matrix from the i
            convolution = np.sum(kernelx*sub_matrix)   #Convolution of each pixel
            empty_matrix[i][j] = convolution           #Stores the convoluted results in

    return empty_matrix
```

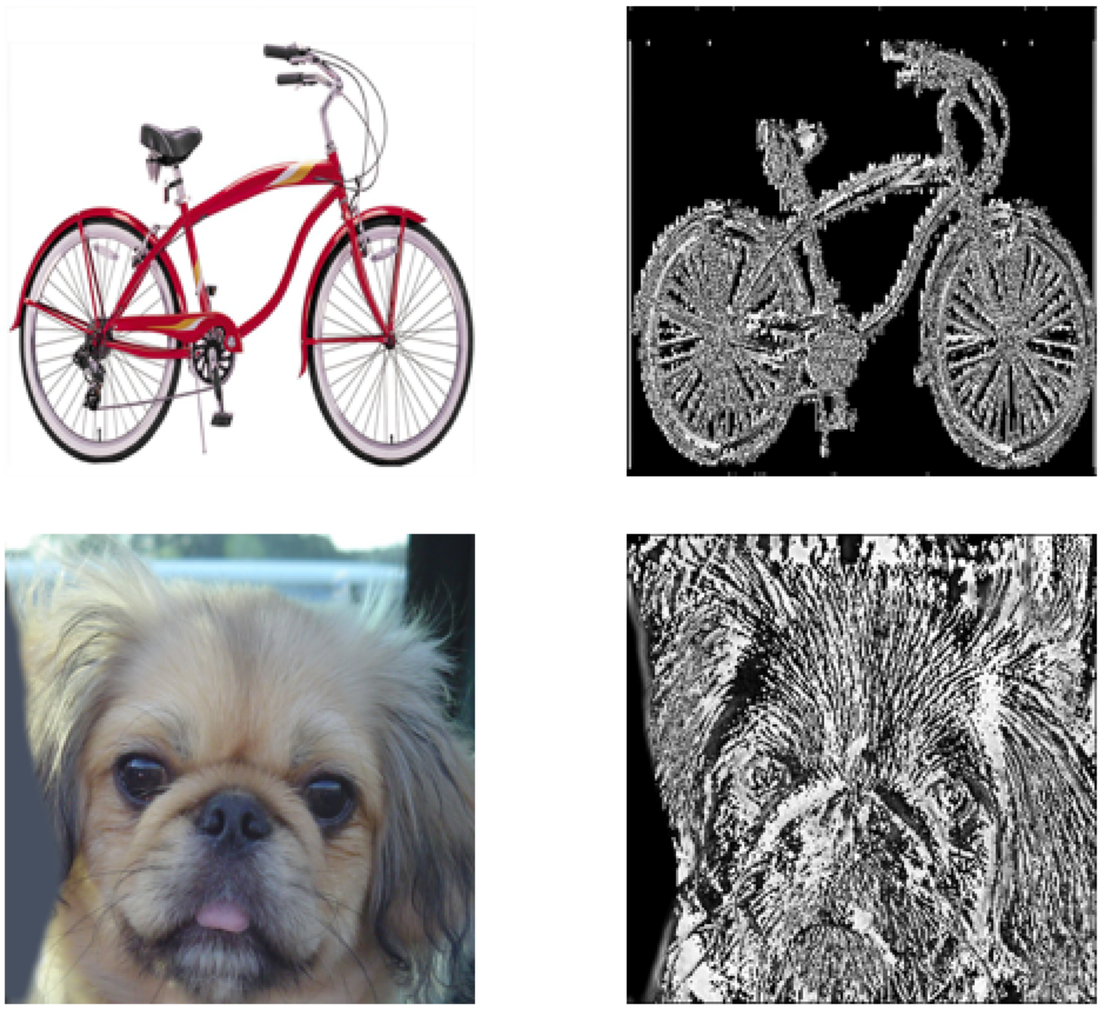

Figure 1.6. Sobel X Filter without OpenCV

1.7 PART 4: SOBEL Y AXIS NO OPENCV

This was identical to the x axis. Here is the code for this function:

```
def sobel_filter_y_command(image_path):
    image_cv2 = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    kernel_matrix = np.array([[-1,-2,-1],[0,0,0],[1,2,1]])
    copy_matrix = image_cv2
    empty_matrix = np.zeros_like(copy_matrix)

    width, height = image_cv2.shape #Stores width and height of image

    for i in range(1, width - 1):
        for j in range(1, height - 1):
            sub_matrix = image_cv2[i-1:i+2, j-1:j+2]
            convolution = np.sum(kernel_matrix*sub_matrix)
            empty_matrix[i][j] = convolution
    return empty_matrix
```

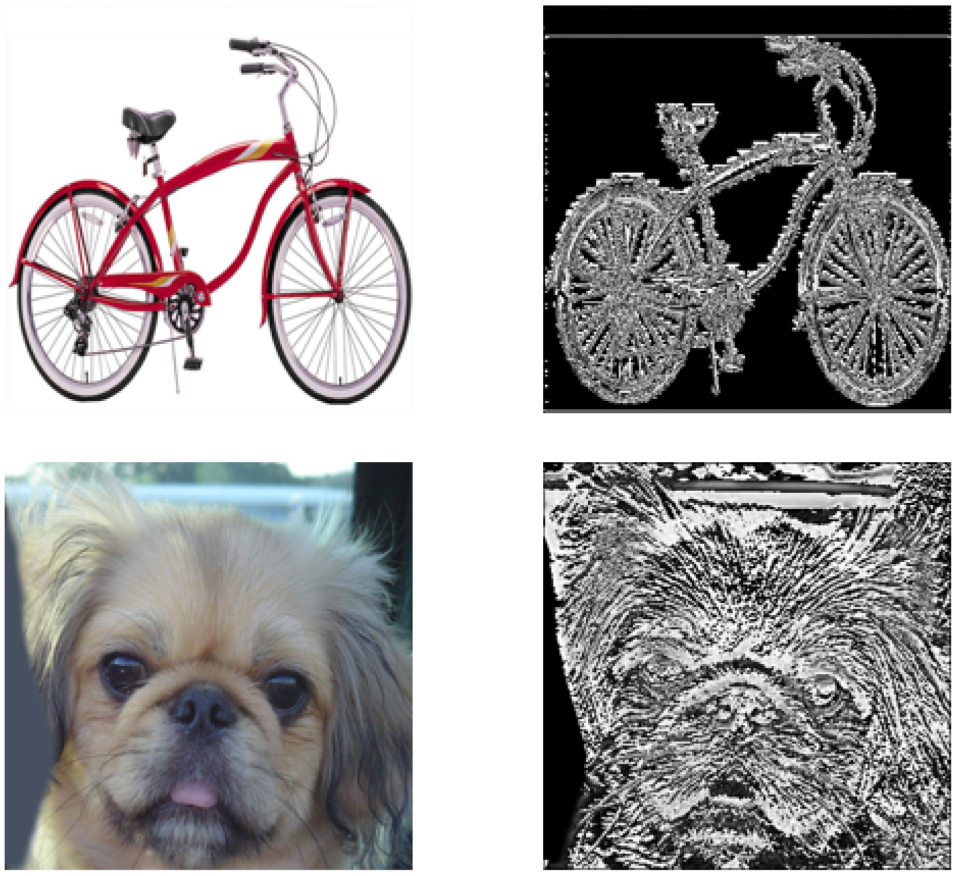

Figure 1.7. Sobel Y Filter without OpenCV

1.8 PART 5: SOBEL X AND Y AXIS NO OPENCV

This was identical to the x and axis code, I called both functions in here and made it work this way however I was having troubles producing a result. This is what the result should look like tho.. Here is the code for this function:

```
def sobel_filter_xy_command(image_path):

    sobelx = sobel_filter_x_command(image_path)
    sobely = sobel_filter_y_command(image_path)
    magnitude = np.sqrt(sobelx**2 + sobely**2)

    return magnitude
```

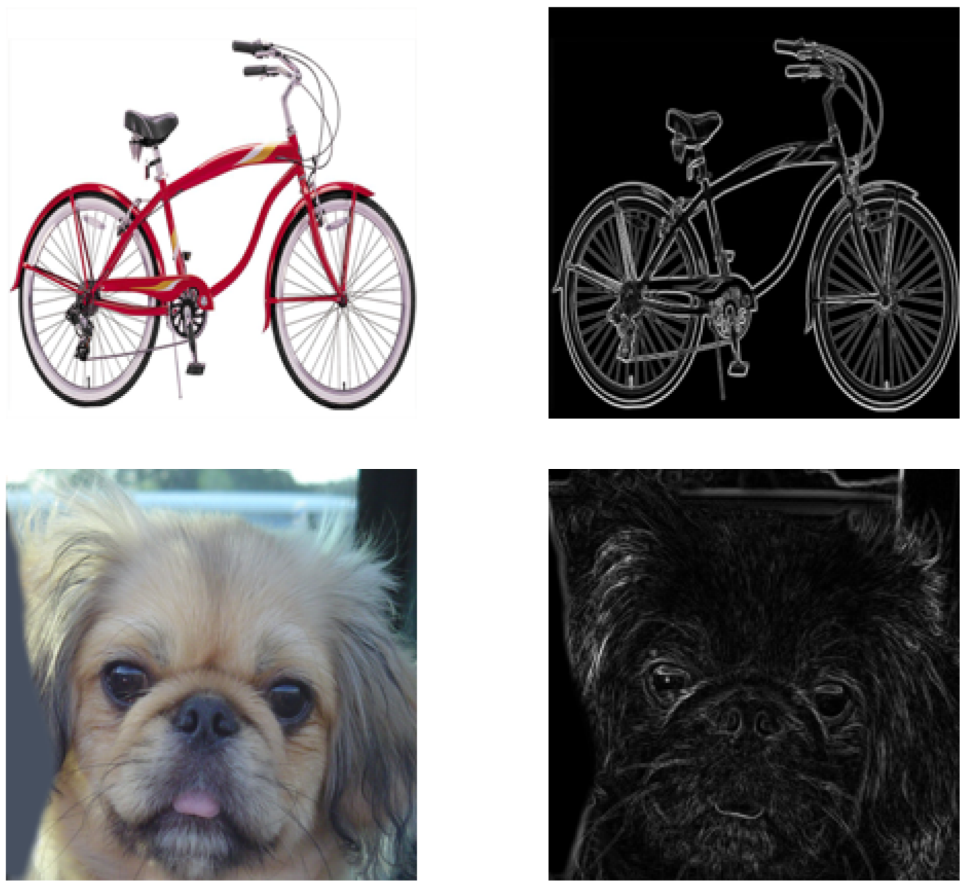

Figure 1.8. Sobel XY Filter without OpenCV

1.9 PART 6: SOBEL X AND Y AXIS OPENCV

This was identical to the x axis. Here is the code for this function:

```
def sobel_filter(image_path, k):
    image_cv2 = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    sobelx = cv2.Sobel(image_cv2, cv2.CV_64F, 1, 0, k)
    sobely = cv2.Sobel(image_cv2, cv2.CV_64F, 0, 1, k)
    sobelxy = cv2.magnitude(sobelx, sobely)
    return sobelxy
```

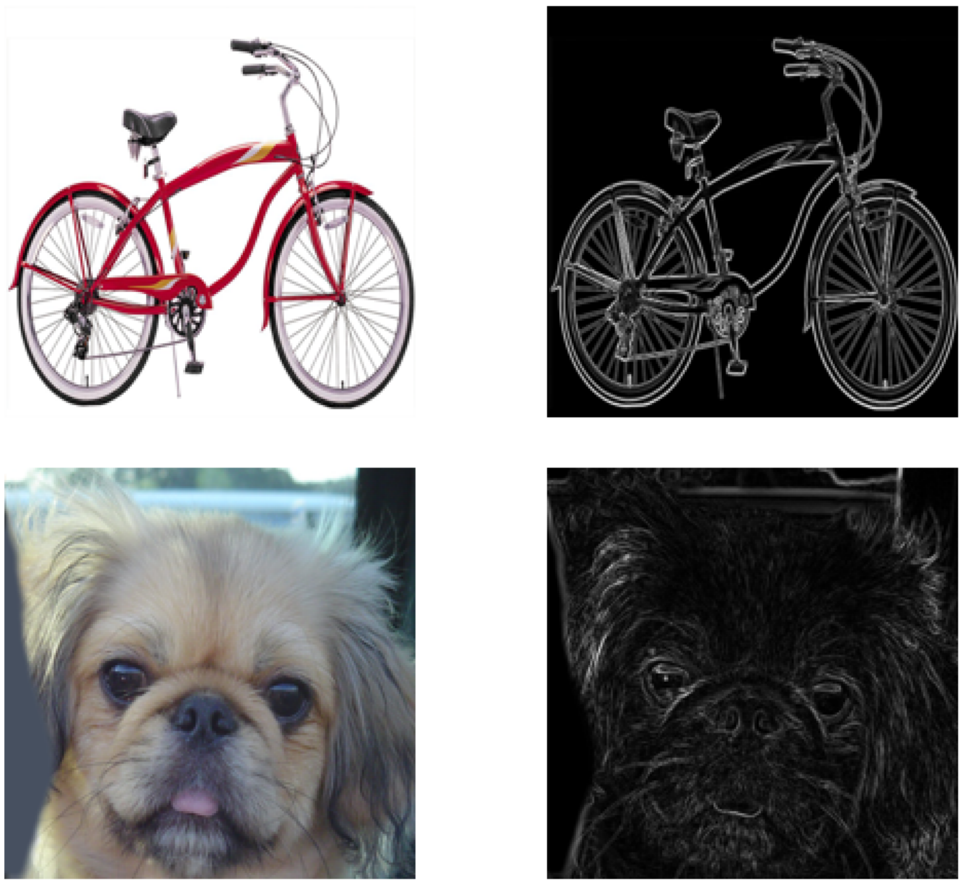

Figure 1.9. Sobel X and Y Filter with OpenCV

1.10 PART 7: GAUSSIAN FILTER OPENCV

This was similar in terms of difficulty for the openCV filters as this became only a few short lines of code. Here is the code for this function:

```
def gaussian_Filter(image_path, k):
    image_cv2 = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    image_cv2 = cv2.GaussianBlur(image_cv2, (k, k), 0)
    return image_cv2
```

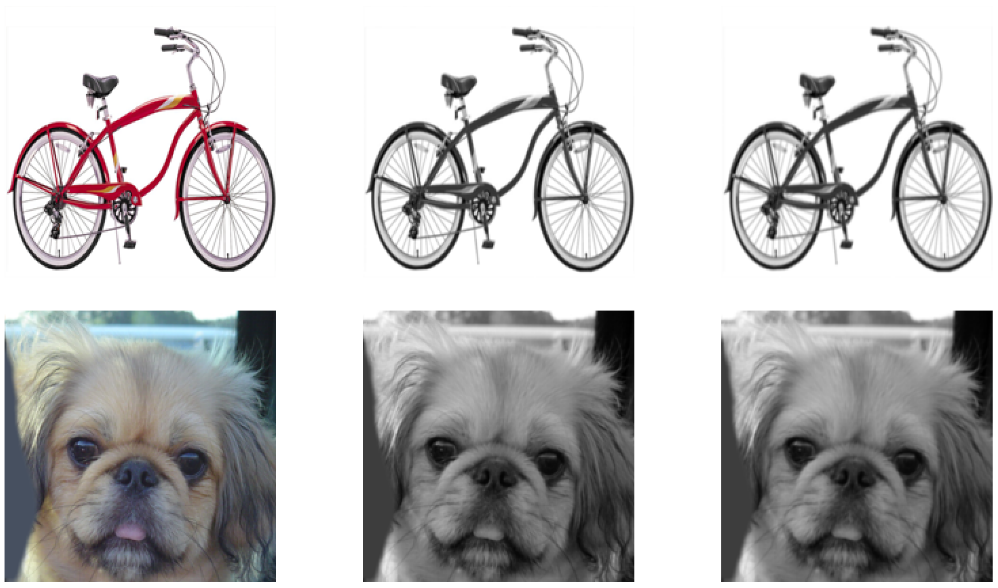

Figure 1.10. Gaussian filter with OpenCV

References

https://www.geeksforgeeks.org/python-opencv-cheat-sheet/
https://www.geeksforgeeks.org/open-a-new-window-with-a-button-in-python-tkinter
https://www.geeksforgeeks.org/opencv-python-tutorial/
https://stackoverflow.com/questions/17815687/image-processing-implementing-sobel-filter
https://numpy.org/doc/stable/reference/generated/numpy.convolve.html
https://www.tutorialkart.com/opencv/python/opencv-python-gaussian-image-smoothing/
https://stackoverflow.com/questions/15485102/how-to-correctly-use-cv-boxfilter-function-in-python
https://www.geeksforgeeks.org/python-program-to-detect-the-edges-of-an-image-using-opencv-sobel-edge-detection/
https://adeveloperdiary.com/data-science/computer-vision/how-to-implement-sobel-edge-detection-using-python-from-scratch/