Assignment #2

Prepared by
Michael Pérez

Computer Vision
CAP 4410

4 October 2019

Florida Polytechnic University

Table of Contents

List of Figures

1. Introduction

1.1 Introduction

My name is Michael Pérez, and I am a senior studying Computer Science at Florida Polytechnic University. This objective of this project was to create and develop seven filters. I implemented these filters using OpenCV in Visual Studio, in C++. I tested each filter on the following images:

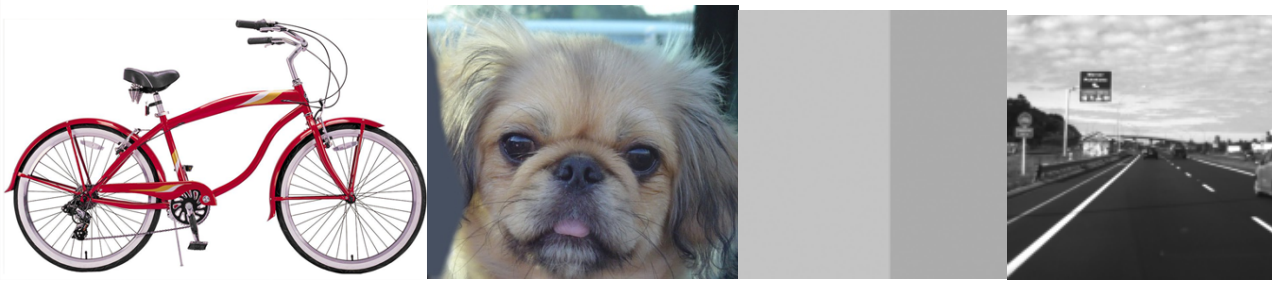

Figure 1: Test Images

1.2 Running the Project

There is one window that appears immediately when the program is run:

1. Image

When a filter is applied to an image, the edited image is displayed in a new window:

2. Edited Image

While the program is running, pressing the following buttons corresponds to these commands:

'1': Apply the box filter that I coded myself and display the edited image.
'2': Apply OpenCV's box filter and display the edited image.
'3': Apply the Sobel filter towards x-axis edges that I coded myself and display the edited image.
'4': Apply the Sobel filter towards y-axis edges that I coded myself and display the edited image.
'5': Apply the Sobel filter towards x-axis and y-axis edges that I coded myself and display the edited image.
'6': Apply OpenCV's Sobel filter towards x-axis and y-axis edges and display the edited image.
'7': Apply OpenCV's Gaussian filter and display the edited image.

1.3 Product Scope

This product has the ability of applying seven filters to an image. These filters can be useful for preprocessing images and videos for other computer vision applications and for photographers. Sobel edge detection only leaves the edges of the image, discarding pixels that are not relevant for

the edges of the image. This is helpful for applications like image segmentation, to decrease the amount of data that needs to be processed. Additionally, the filters can be helpful to a photographer who would like to smoothen an image.

2. Box Filter (coded from scratch)

2.1 Implementation

I read the textbook [1] and created an algorithm that creates a kernel matrix the same size as the window size (3x3, 5x5, or 7x7).

```
if (windowSize == 3)
{
	double kernel[3][3] = {{1 / 9.0, 1 / 9.0, 1 / 9.0},
	                       {1 / 9.0, 1 / 9.0, 1 / 9.0},
	                       {1 / 9.0, 1 / 9.0, 1 / 9.0}};

	editedImage = applyKernel(image, kernel);
	imshow("Edited Image", editedImage);
}
else if (windowSize == 5)
{
	double kernel[5][5] = {{1 / 25.0, 1 / 25.0, 1 / 25.0, 1 / 25.0, 1 / 25.0},
	                       {1 / 25.0, 1 / 25.0, 1 / 25.0, 1 / 25.0, 1 / 25.0},
	                       {1 / 25.0, 1 / 25.0, 1 / 25.0, 1 / 25.0, 1 / 25.0},
	                       {1 / 25.0, 1 / 25.0, 1 / 25.0, 1 / 25.0, 1 / 25.0},
	                       {1 / 25.0, 1 / 25.0, 1 / 25.0, 1 / 25.0, 1 / 25.0}};
	editedImage = applyKernel(image, kernel);
	imshow("Edited Image", editedImage);
}
else if (windowSize == 7)
{
	double kernel[7][7] = {{1 / 49.0, 1 / 49.0, 1 / 49.0, 1 / 49.0, 1 / 49.0,  1 / 49.0,  1 / 49.0},
	                       {1 / 49.0, 1 / 49.0, 1 / 49.0, 1 / 49.0, 1 / 49.0,  1 / 49.0,  1 / 49.0},
	                       {1 / 49.0, 1 / 49.0, 1 / 49.0, 1 / 49.0, 1 / 49.0,  1 / 49.0,  1 / 49.0},
	                       {1 / 49.0, 1 / 49.0, 1 / 49.0, 1 / 49.0, 1 / 49.0,  1 / 49.0,  1 / 49.0},
	                       {1 / 49.0, 1 / 49.0, 1 / 49.0, 1 / 49.0, 1 / 49.0,  1 / 49.0,  1 / 49.0},
	                       {1 / 49.0, 1 / 49.0, 1 / 49.0, 1 / 49.0, 1 / 49.0,  1 / 49.0,  1 / 49.0},
	                       {1 / 49.0, 1 / 49.0, 1 / 49.0, 1 / 49.0, 1 / 49.0,  1 / 49.0,  1 / 49.0}};
	editedImage = applyKernel(image, kernel);
	imshow("Edited Image", editedImage);
}
```

Figure 2: Box Filter

Then, I call the function applyKernel(), which takes the input image and the kernel as parameters, and performs convolution. There are three functions named applyKernel(), and each has a different sized kernel as a parameter (3x3, 5x5, and 7x7). This function loops through each pixel in the matrix, then loops through each pixel in the window. Inside that loop, each pixel in the window is multiplied by the corresponding entry in the kernel matrix, for each color channel. I made sure that depending on the window size, the loops do not access any invalid matrix entries (i.e. kernelFilter[-1][-1] would result in an error). Then, pixel intensity values less than 0 and greater

than 255 are capped at 0 and 255, respectively. The window's center pixel is then assigned the resultant RGB values. The convoluted image is returned to the main function and the edited image is displayed.

```
Mat applyKernel(Mat image, double kernelFilter[3][3])
{
	Mat result;
	image.copyTo(result);

	for (int y = 1; y < image.rows - 2; y++)
	{
		for (int x = 1; x < image.cols - 2; x++)
		{
			red = 0.0, green = 0.0, blue = 0.0;
			for (int k = -1; k <= 1; k++)
			{
				for (int j = -1; j <= 1; j++)
				{
					red += image.at<Vec3b>(y + j, x + k)[0] * kernelFilter[j + 1][k + 1];
					green += image.at<Vec3b>(y + j, x + k)[1] * kernelFilter[j + 1][k + 1];
					blue += image.at<Vec3b>(y + j, x + k)[2] * kernelFilter[j + 1][k + 1];
				}
			}

			if (red > 255)
				red = 255;
			else if (red < 0)
				red = 0;
			if (green > 255)
				green = 255;
			else if (green < 0)
				green = 0;
			if (blue > 255)
				blue = 255;
			else if (blue < 0)
				blue = 0;

			result.at<Vec3b>(y, x)[0] = red;
			result.at<Vec3b>(y, x)[1] = green;
			result.at<Vec3b>(y, x)[2] = blue;
		}
	}
	return result;
}
```

Figure 3: applyKernel()

2.2 Testing

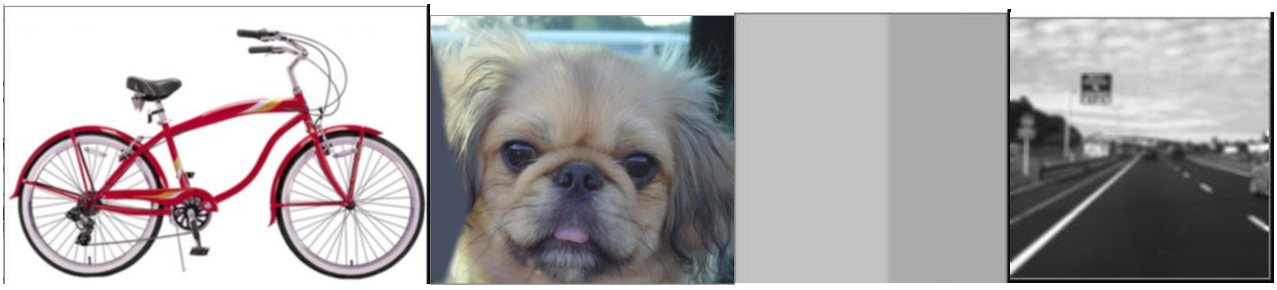

Figure 4: Box Filter (coded from scratch), Window Size = 3

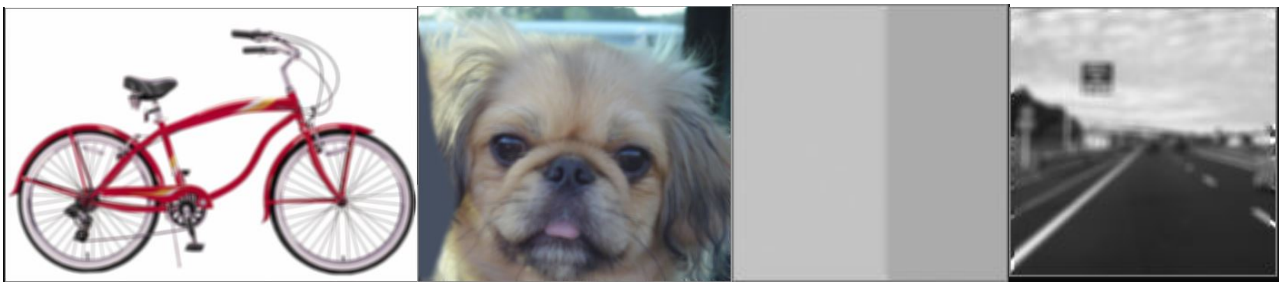

Figure 5: Box Filter (coded from scratch), Window Size = 5

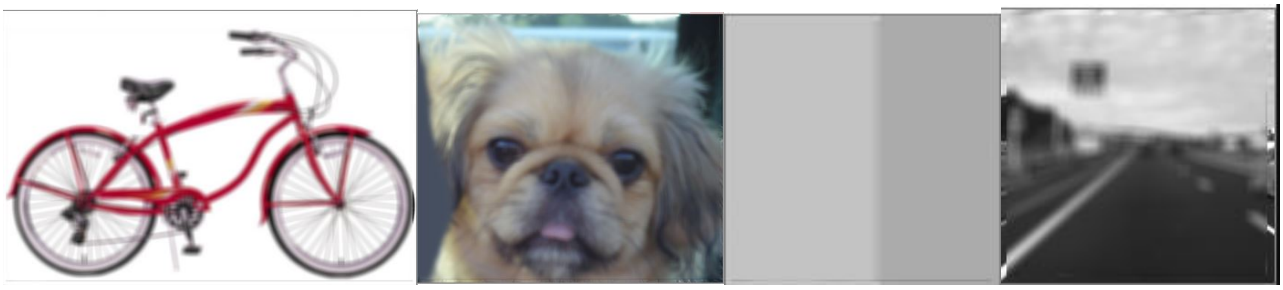

Figure 6: Box Filter (coded from scratch), Window Size = 7

2.3 Discussion

As the window size increases, the box filter's blurring effect increases. Since I did not use a border pixel strategy, when the filter is applied, the number of rows and the number of columns both decrease by one. The filter works as intended.

3. Box Filter (OpenCV)

3.1 Implementation

I researched online [2] how to implement OpenCV's built-in Box Filter method, and implemented this filter with these two lines of code.

```
boxFilter(image, editedImage, -1, Size(windowSize,windowSize), Point(-1,-1), true, BORDER_DEFAULT);
imshow("Edited Image", editedImage);
```

Figure 7: Box Filter (OpenCV)

3.2 Testing

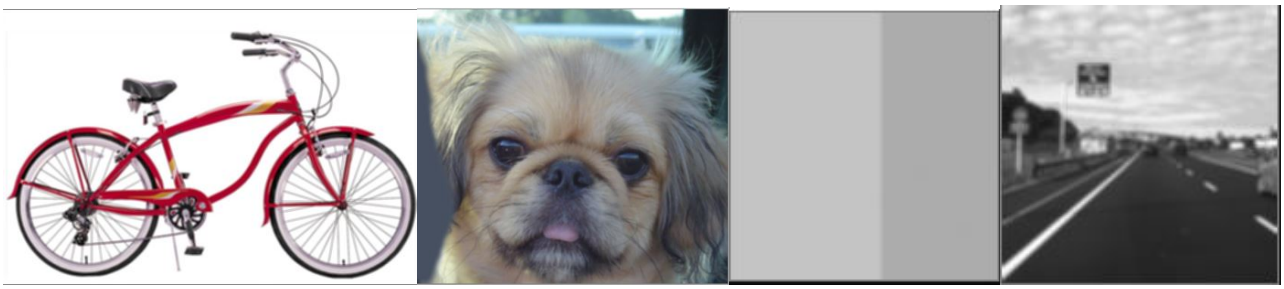

Figure 8: Box Filter (OpenCV), Window Size = 3

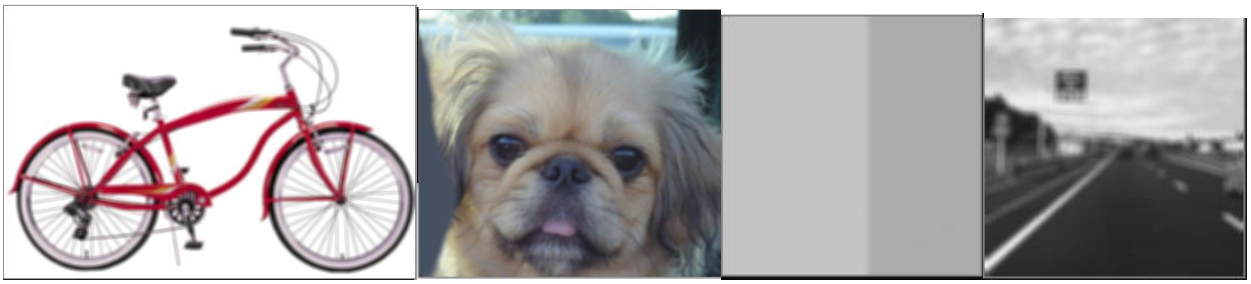

Figure 9: Box Filter (OpenCV), Window Size = 5

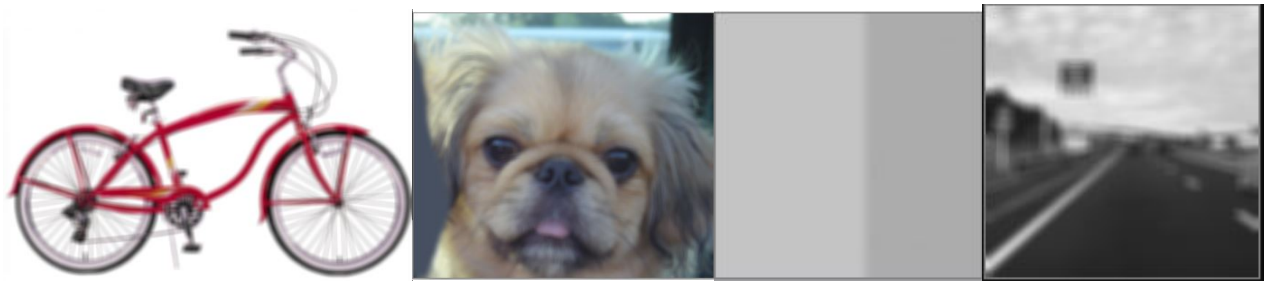

Figure 10: Box Filter (OpenCV), Window Size = 7

3.3 Discussion

As the window size increases, the box filter's blurring effect increases, just like in the Box Filter I created. I cannot notice any difference in the effect of my Box Filter and OpenCV's Box Filter on these images. The number of rows and the number of columns do not decrease by one in this filter, because OpenCV's Box Filter uses a border pixel strategy. The filter works as intended.

4. Sobel Filter towards x-axis edges (coded from scratch)

4.1 Implementation

I implemented this filter similarly to how I implemented the Box Filter. I created a 3x3, 5x5, or 7x7 Sobel filter kernel matrix that finds vertical edges. Then, I applied convolution to the image with the kernel using the same applyKernel() function described above.

```
if (windowSize == 3)
{
        double kernel[3][3] = {{-1, -2, -1},
                               {0, 0, 0},
                               {1, 2, 1}};

        editedImage = applyKernel(image, kernel);
        imshow("Edited Image", editedImage);
}
```

Figure 11: Sobel Filter, horizontal edges, coded from scratch

4.2 Testing

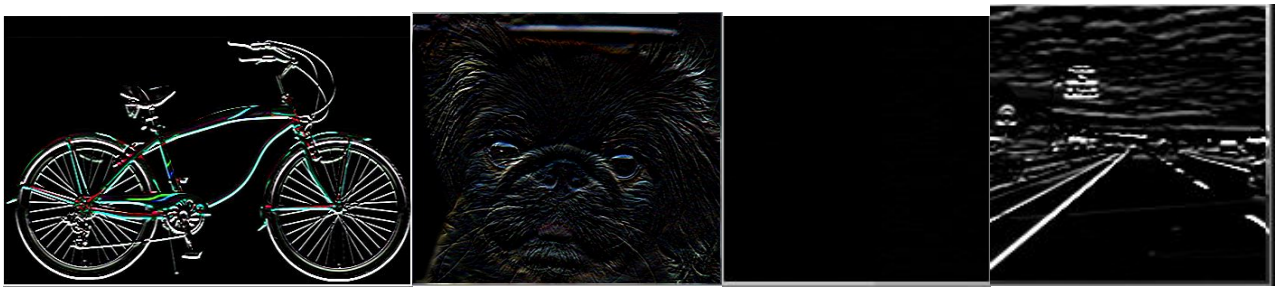

Figure 12: Sobel Filter, horizontal edges, coded from scratch, Window Size = 3

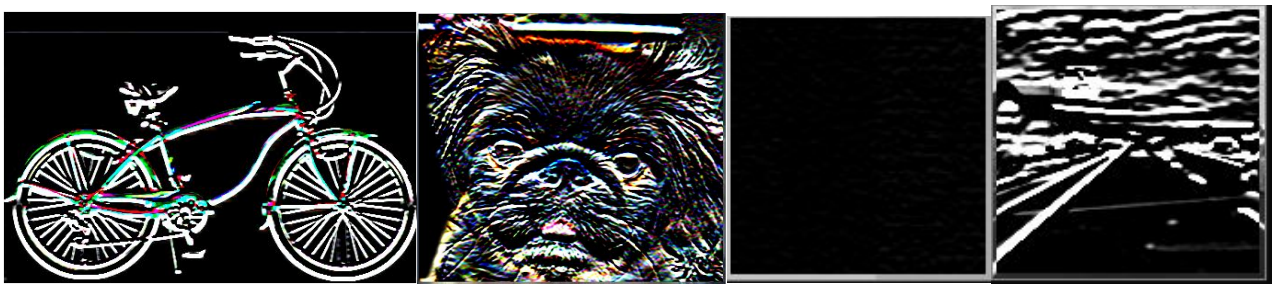

Figure 13: Sobel Filter, horizontal edges, coded from scratch, Window Size = 5

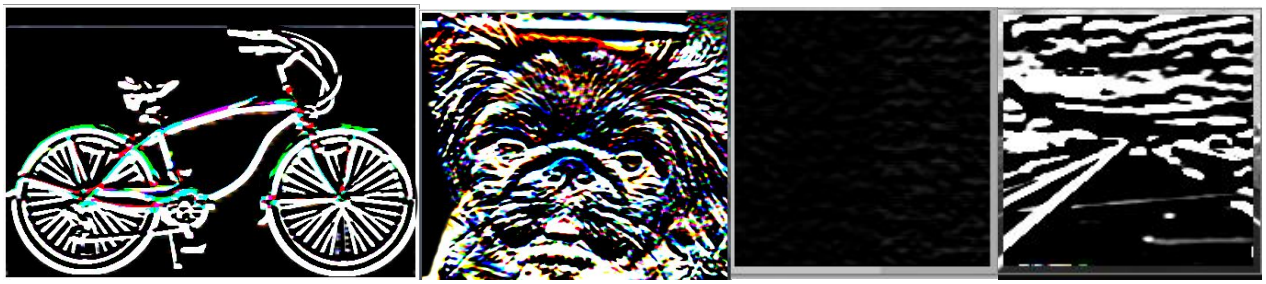

Figure 14: Sobel Filter, horizontal edges, coded from scratch, Window Size = 7

4.3 Discussion

I designed this filter to work with color images and to output the transformed image in color as well. As the window size increases, the vertical edge detection increases. The edge in image #3 is not detected for any window size because this filter finds horizontal edges, not vertical ones.

5. Sobel Filter towards y-axis edges (coded from scratch)

5.1 Implementation

I implemented this filter similarly to how I implemented the other two filters that I coded from scratch. I created a 3x3, 5x5, or 7x7 Sobel filter kernel matrix that finds horizontal edges. Then, I applied convolution to the image with the kernel using the same applyKernel() function described above.

```
double kernel[3][3] = {
{-1, 0, 1},
{-2, 0, 2},
{-1, 0, 1}};

editedImage = applyKernel(image, kernel);
imshow("Edited Image", editedImage);
```

Figure 15: Sobel Filter, vertical edges, coded from scratch

5.2 Testing

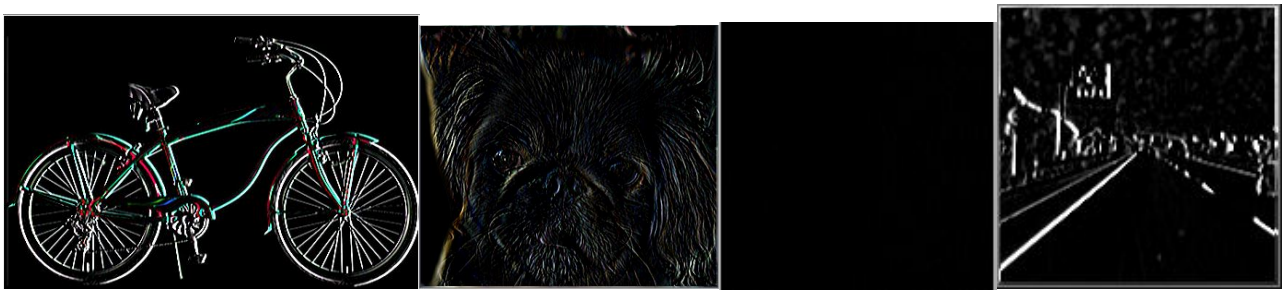

Figure 16: Sobel Filter, horizontal edges, coded from scratch, Window Size = 3

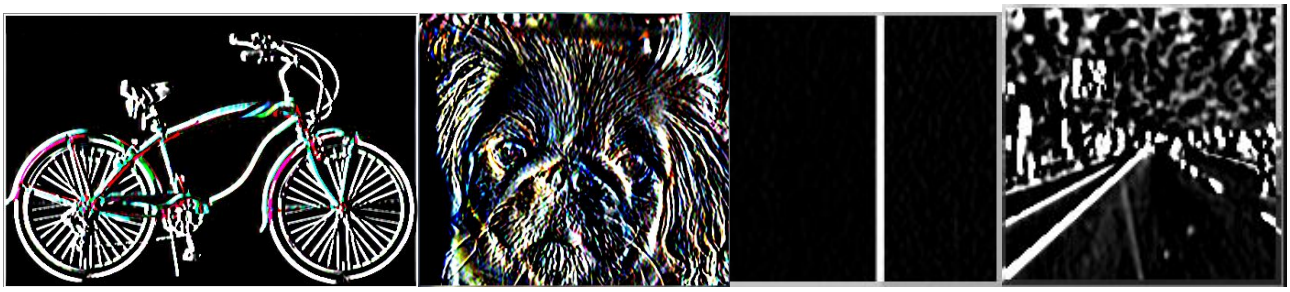

Figure 17: Sobel Filter, vertical edges, coded from scratch, Window Size = 5

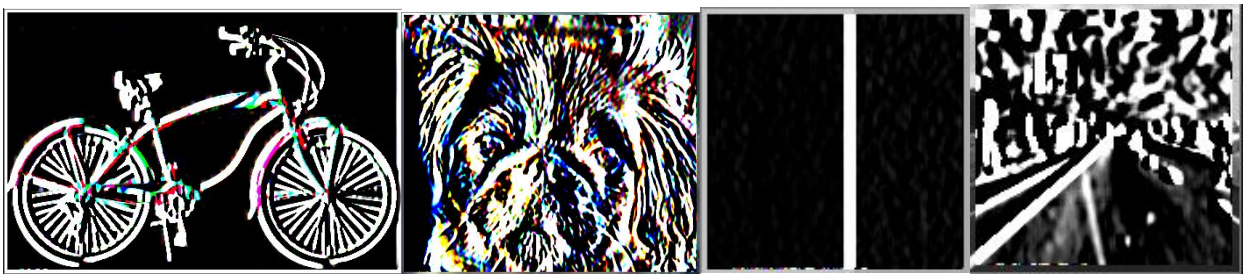

Figure 18: Sobel Filter, vertical edges, coded from scratch, Window Size = 7

5.3 Discussion

I designed this filter to work with color images and to output the transformed image in color as well. As the window size increases, the horizontal edge detection increases. The edge in image #3 is not detected when the window size is three pixels. I suspect this is happening because the edge in the original image is a slight difference in grayscale intensity, which the algorithm cannot detect with a small window size.

6. Sobel Filter towards x-axis and y-axis edges (coded from scratch)

6.1 Implementation

I implemented this filter by first applying the last two filters to the image: Sobel horizontal edge detector and vertical edge detector. Next, I combined the two resultant images into one image in which vertical edges and horizontal edges are detected. For each pixel in the final image, I computed the gradient of the two pixel intensities in each image, for each channel. I capped the maximum and minimum intensity values, like I did in the applyFilter() function, then assigned the pixel RGB values.

```
if (windowSize == 3)
{
	image.copyTo(editedImage);
	double kernel1[3][3] = {{-1, -2, -1},
	                        {0, 0, 0},
	                        {1, 2, 1}};

	double kernel2[3][3] = {{-1, 0, 1},
	                        {-2, 0, 2},
	                        {-1, 0, 1}};

	imageX = applyKernel(image, kernel1);
	imageY = applyKernel(image, kernel2);

	for (int y = 1; y < image.rows - 2; y++)
	{
		for (int x = 1; x < image.cols - 2; x++)
		{
			red = sqrt(pow(imageX.at<Vec3b>(y, x)[0], 2) + pow(imageY.at<Vec3b>(y, x)[0], 2));
			green = sqrt(pow(imageX.at<Vec3b>(y, x)[1], 2) + pow(imageY.at<Vec3b>(y, x)[1], 2));
			blue = sqrt(pow(imageX.at<Vec3b>(y, x)[2], 2) + pow(imageY.at<Vec3b>(y, x)[2], 2));

			if (red > 255)
				red = 255;
			else if (red < 0)
				red = 0;
			if (green > 255)
				green = 255;
			else if (green < 0)
				green = 0;
			if (blue > 255)
				blue = 255;
			else if (blue < 0)
				blue = 0;

			editedImage.at<Vec3b>(y, x)[0] = red;
			editedImage.at<Vec3b>(y, x)[1] = green;
			editedImage.at<Vec3b>(y, x)[2] = blue;
		}
	}
	imshow("Edited Image", editedImage);
}
```

Figure 19: Sobel Filter, horizontal and vertical edges, coded from scratch

6.2 Testing

Window size = 3:

Figure 20: Sobel Filter, horizontal and vertical edges, coded from scratch, Window Size = 3

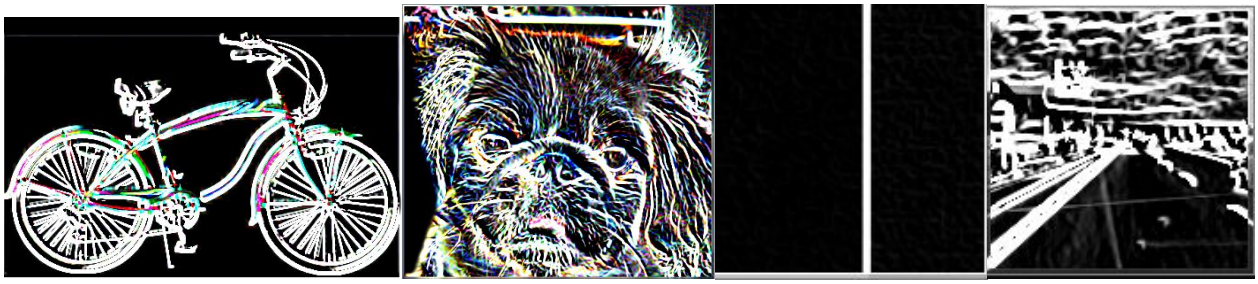

Figure 21: Sobel Filter, horizontal and vertical edges, coded from scratch, Window Size = 5

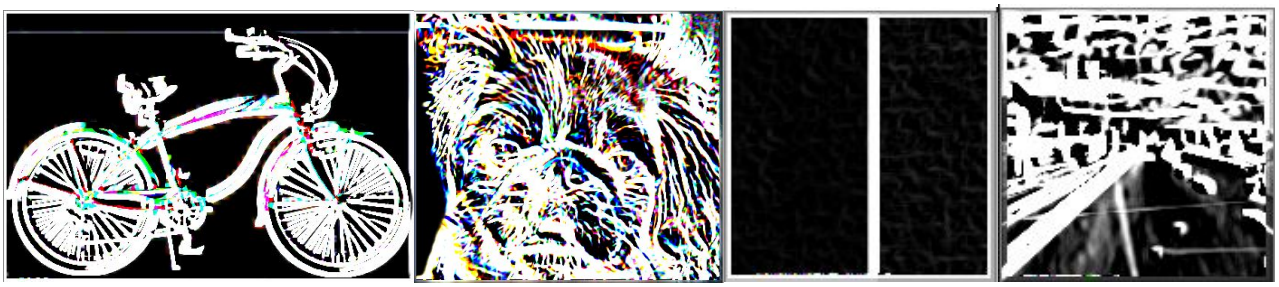

Figure 22: Sobel Filter, horizontal and vertical edges, coded from scratch, Window Size = 7

6.3 Discussion

This filter works best when the window size is three. For window size five and seven, I created the Sobel matrix kernel from sources I found on the internet [3], which were not very reputable. That is why the Sobel edge detector filter did not come out well for window size five and seven.

7. Sobel Filter towards x-axis and y-axis edges (OpenCV)

7.1 Implementation

I researched online [4] how to implement OpenCV's built-in Gaussian Filter method, and implemented it by first converting the image to grayscale, since the Sobel function only accepts grayscale images as a parameter. Then the Sobel functions create the derivative of the image in the x and y directions. The final lines of code compute the sum of the absolute value of the pixels in each image, and caps the maximum and minimum intensity values, then displays the edited image.

```
// Convert the image to grayscale
cvtColor(image, grayImage, COLOR_BGR2GRAY);

// Generate grad_x and grad_y
Mat grad_x, grad_y;

// Gradient X
Sobel(grayImage, imageX, CV_16S, 1, 0, windowSize, 10, 10, BORDER_DEFAULT);

// Gradient Y
Sobel(grayImage, imageY, CV_16S, 0, 1, windowSize, 10, 10, BORDER_DEFAULT);

sobel = abs(imageX) + abs(imageY);

minMaxLoc(sobel, &sobmin, &sobmax);
sobel.convertTo(editedImage, CV_8U, 255. / sobmax, 0);

imshow("Edited Image", editedImage);
```

Figure 23: Sobel Filter, horizontal and vertical edges (OpenCV)

7.2 Testing

Figure 24: Sobel Filter, horizontal and vertical edges (OpenCV), Window Size = 3

Figure 25: Sobel Filter, horizontal and vertical edges (OpenCV), Window Size = 5

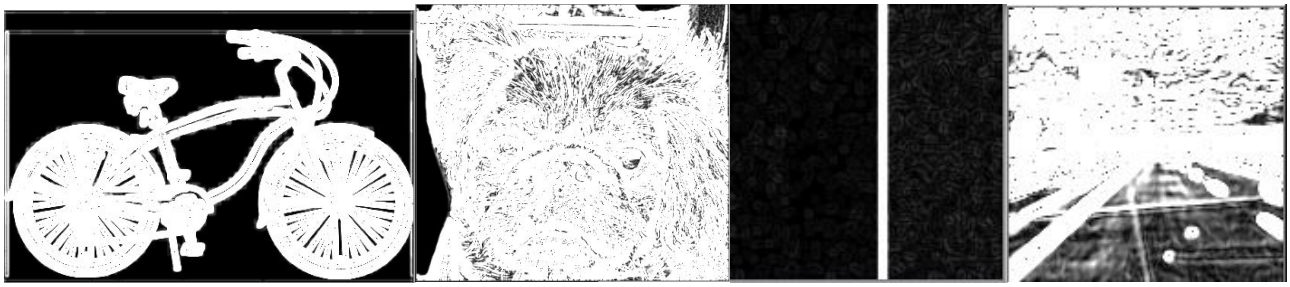

Figure 26: Sobel Filter, horizontal and vertical edges (OpenCV), Window Size = 7

7.3 Discussion

This filter works best when the window size is three or five. When the window size is seven, the edge detection is too high, and unwanted noise appears. Other than this, the filter works as intended.

8. Gaussian Filter (OpenCV)

8.1 Implementation

I researched online [2] how to implement OpenCV's built-in Gaussian Filter method, and implemented it with these two lines of code:

```
GaussianBlur(image, editedImage, Size(windowSize, windowSize), 0, 0, BORDER_DEFAULT);
imshow("Edited Image", editedImage);
```

Figure 27: Gaussian Filter (OpenCV)

8.2 Testing

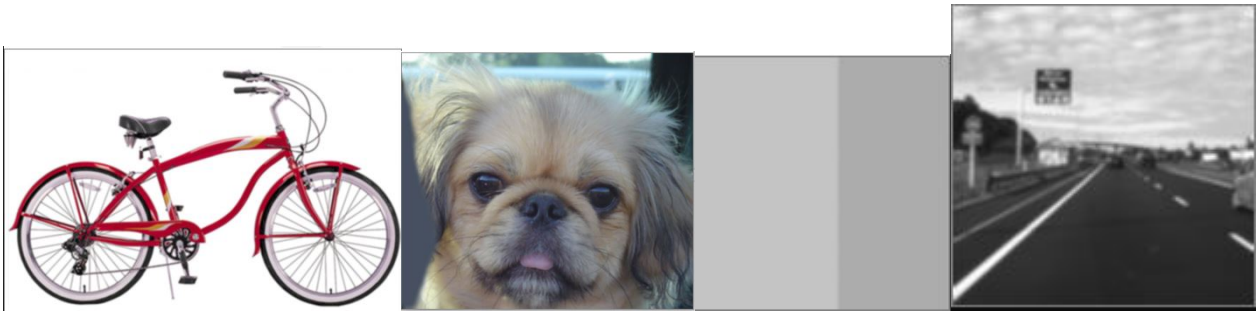

Figure 28: Gaussian Filter (OpenCV), Window Size = 3

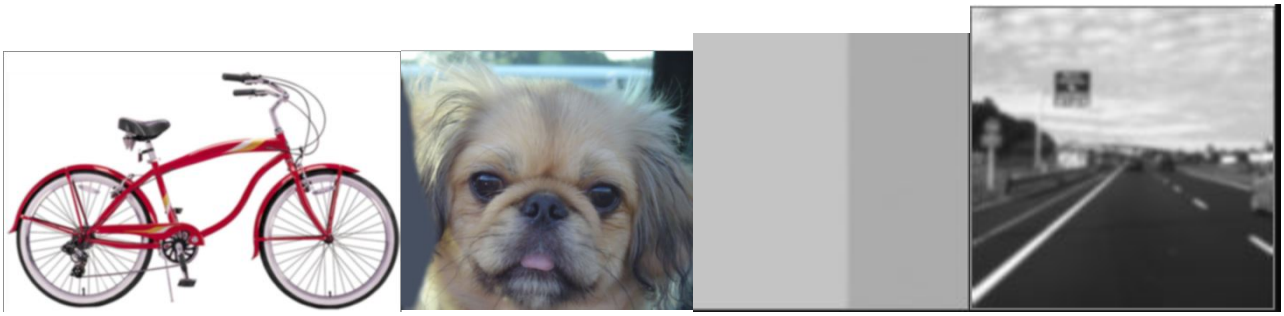

Figure 29: Gaussian Filter (OpenCV), Window Size = 5

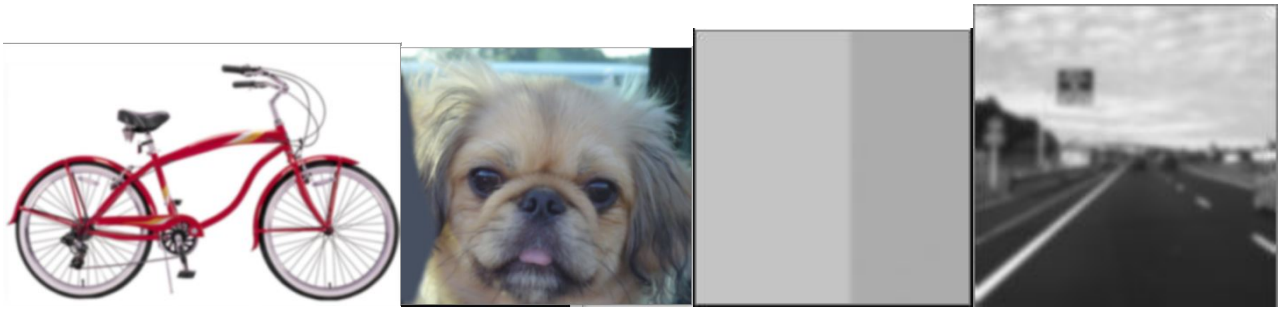

Figure 30: Gaussian Filter (OpenCV), Window Size = 7

8.3 Discussion

As the window size increases, the Gaussian Filter's blurring effect also increases. The results of this filter are similar to the results of the Box Filters; a general smoothing effect is applied to the image. The filter works as intended.

References

[1] R. Laganiere, *OpenCV Computer Vision Application Programming Cookbook*. Birmingham, U.K.: Packt Pub., 2014.
[2] "Image Filtering — OpenCV 2.4.13.7 documentation", *Docs.opencv.org*, 2019. [Online]. Available: https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html. [Accessed: 05- Oct- 2019].
[3] P. R and A. Bowen, "Sobel filter kernel of large size", *Stack Overflow*, 2019. [Online]. Available: https://stackoverflow.com/questions/9567882/sobel-filter-kernel-of-large-size. [Accessed: 05- Oct- 2019].
[4] "OpenCV: Sobel Derivatives", *Docs.opencv.org*, 2019. [Online]. Available: https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html. [Accessed: 05- Oct- 2019].