Exploring Canny Edge Detection in Python

Introduction to Computer Vision - University of Nebraska

Zachary Swanson
March 01, 2024

Contents

1 Introduction

Edge detection is a computational method of isolating regions of sudden changes (i.e. edges) in a digital image. This is an important algorithm because edges provide significant identifying information about the objects in the image. Hence, the detection of edges is a fundamental algorithm that enables the development of more advanced computer vision concepts.

There are multiple methods for detecting edges and the most common is arguably the application of derivative kernels to a digital image. However, as highlighted in [1], noisy images (an ever-present reality of digital images) cause simple kernel operations to produce edges that are too thick, missing, and extraneous. Yet, there are techniques that improve the detected edges despite the limitations of simple kernel operations.

The "Canny edge detector" and "facet model" are presented in [1] as two significant and popular methods for identifying edges more precisely. The Canny edge detector was presented by John Canny in his 1986 publication [2] and implementation of Canny's algorithm is the focus of this project. The Canny edge detector is a multi-step algorithm that combines other algorithms like derivative kernels to produce contiguous, proper width edges while emphasizing strong edges (further details in Background). Therefore, practical implementation of such an important edge detection is fundamental for the development of students exploring computer vision. The purpose of this project is to develop a functional computer program that implements the Canny edge detector with user-configurable parameters relating to the algorithm. The parameters provide a means for exploring the extents of the algorithm and understanding the impact of various key functions within the algorithm.

2 Background

The Canny edge detector is a multi-part algorithm with each part contributing to the goal of improved edge detection. Figure 1 groups related parts by color and illustrates the flow of the algorithm as presented in 1. The following subsections will briefly describe the groups and their importance to the overall algorithm.

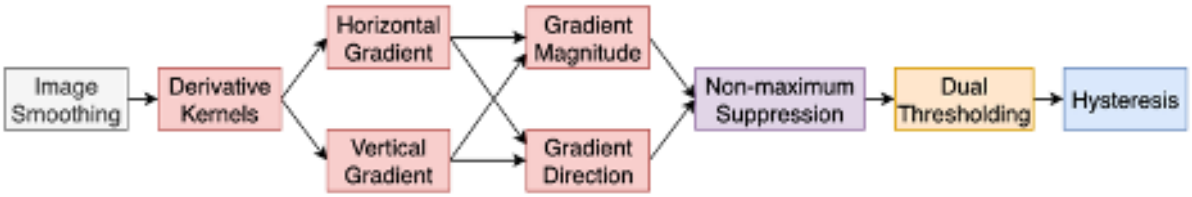

Figure 1: High-level diagram of the steps comprising the Canny edge detection algorithm

2.1 Image Smoothing

Noise presents a problem for edge detection because noise introduces additional sudden intensity changes in the digital image. The sudden change of intensity due to noise and due to a true edge are indistinguishable for naive edge detectors. Smoothing techniques can alleviate some effects of noise in the image and may be accomplished by application of a Gaussian kernel to the digital image.

In [2], Canny derived the optimal operators for detecting various edge types (e.g. step, roof, etc.) and found the best approximations to be derivatives of the Gaussian function. The

Gaussian function in one dimension is

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{1}$$

where μ is the mean and σ is standard deviation. The benefit of the Gaussian approximation was computational efficiency compared to the optimal operator. Canny also stated that due to the associativity of convolution, the standard Gaussian could be applied prior to the convolution. Hence, the first step in the algorithm is the smoothing of the image by application of a Gaussian kernel. The level of smoothing is controlled by the value of σ and the size of the Gaussian kernel. These are configurable parameters in the Python implementation.

2.2 Gradients

The detection of edges is accomplished by the application of derivative kernels. Application of the kernel is accomplished by "sliding" the kernel across the image left-to-right and top-to-bottom, computing a sum of products at each step. If each step is one-pixel and boundary conditions are appropriately handled the operation will produce an image of equal dimensions to the original. Three derivative kernels were evaluated for this experiment: Prewitt, Sobel, and Scharr. These 3x3 kernels for computing derivatives in the x-direction (y is transposed) are provided as

$$\frac{1}{6}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad \frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \frac{1}{32}\begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix}, \tag{2}$$

respectively.

These one-dimensional provide edges but only as the edges occur perpendicular to the direction of the derivative (e.g. derivatives in the x-direction will detect vertical lines but not horizontal). Furthermore, diagonal edges may appear weak in both x- and y-derivatives. Generating a more comprehensive detection of the edges requires calculation of the gradient magnitude and direction at each pixel. These calculation are, respectively,

$$M(x, y) = |\nabla f| = \sqrt{f_x^2 + f_y^2} \tag{3}$$

and

$$\theta(x, y) = \angle \nabla f = \arctan \frac{f_y}{f_x}, \tag{4}$$

where f_x and f_y are the derivatives in the x- and y-direction respectively [1]. However, these are not necessarily clean edges as they may be too thick, missing, and extraneous in various locations throughout the image.

2.3 Non-maximum Suppression

Non-maximum suppression (NMS) is a method of "cleaning" the edges provided by the gradient magnitude by only keeping edges that are local maxima. As described in [1], NMS is accomplished in the most simple sense by looking one or more pixels in the direction and opposite direction of the gradient. If the current pixel is the maximum of all its neighbors, then it retains it's magnitude value. If not, the magnitude is set to zero. In this manner, detected edges should only be one pixel wide.

2.4 Dual Thresholding and Hysteresis

The edges retain varying levels of magnitude after NMS. The final goal of the Canny edge detector is to isolate "strong" edges. This is accomplished by dual thresholds where an upper and lower threshold are defined. Anything below the lower threshold is considered a non-edge and anything above the upper threshold is considered a strong edge. Weak edges exist between the two thresholds. Strong edges are automatically included in the final edge presentation but weak edges may or may not be included. Hysteresis is applied to determine if weak edges should be included in the final edges. Figure 2 provides a conceptual illustration of how the hysteresis is applied after dual thresholding.

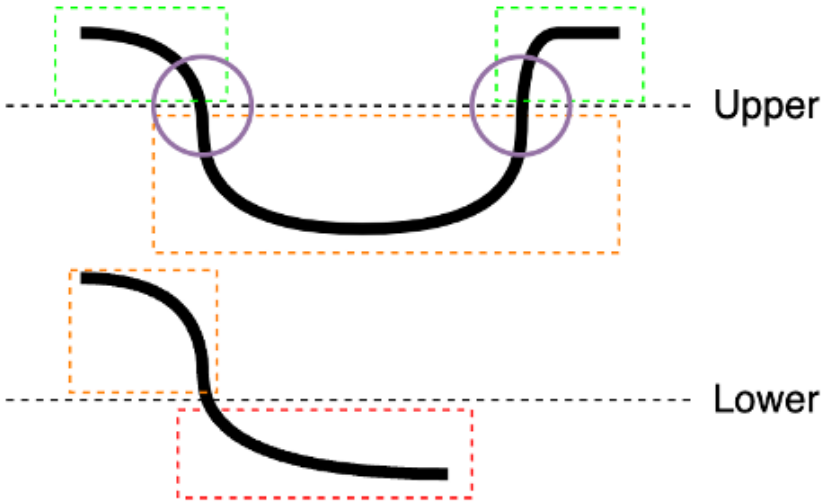

Figure 2: Visual aid demonstrating hysteresis for evaluating weak pixels. Green boxes identify strong pixels; orange boxes identify weak edges; red boxes identify non edges; purple circles identify weak-strong connections (adapted from [3]).

The locations of strong pixels (green) are compared to the locations of weak pixels (orange). If the a strong and a weak pixel are neighbors, then the weak pixel is promoted to strong. This process is repeated until no more weak-strong neighbors exists. Thus, the final number of edges should be larger than the number of edges detected by dual thresholds alone.

2.5 Evaluation Techniques

A review of recent publications related to edge detection was provided in [4] that analyzed the performance evaluation techniques used in the publications. Visual evaluation was the most common technique, where researchers provide original and edge images to discuss the performance of their edge detector. This provides a strong qualitative analysis because humans are capable of looking at image and detecting edges and comparing the perceived edges to the output of the edge detection program.

Several methods were also identified in [4] for quantitative analysis of performance. Many of these techniques required ground truth data. The most popular quantitative metrics found in [4] were based on equations derived from the confusion matrix (illustrated in Figure 3). As shown, the confusion matrix compares the ground truth values of a binary class (e.g. edge versus non-edge) with the predicted values of some algorithm.

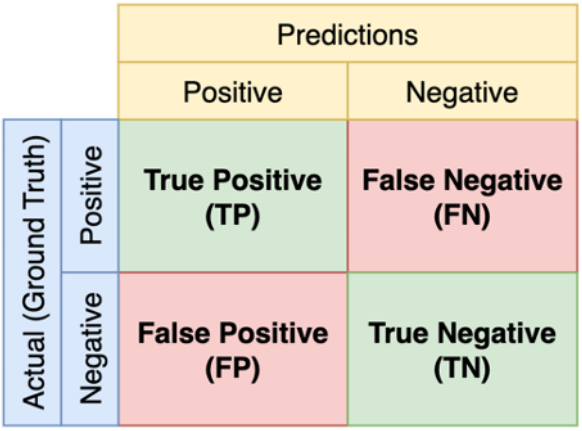

Figure 3: Representation of the confusion matrix comparing ground truth and predicted values.

A multitude of metrics can be derived based on various ratios of the quadrants of the confusion matrix shown in Figure 3. However, the true positive rate (TPR) and the false positive rate (FPR) will be sufficient for the purposes of this experiment. The TPR provides a ratio of the number of correctly predicted or true positive samples (TP) with respect to the total number of actual (ground truth) positive samples (P). Similarly, FPR provides a ratio of the number of incorrectly predicted or false positive samples (FP) with respect to the total number of actual negative samples (N). These may be represented as

$$TPR = \frac{TP}{P} \tag{5}$$

and

$$FPR = \frac{FP}{N}, \tag{6}$$

respectively. With respect to edges, TPR represents how often the edge detector correctly identified edges as edges and FPR represents how often the edge detector incorrectly identified non-edges as edges. A successful edge detector would have high TPR and low FPR.

3 Methodology

3.1 Python Implementation

The Canny edge detector (further referred to as canny-ecen898) was developed as a Python class using the NumPy library for implementation of the necessary matrix operations. The class was initialized with a NumPy array representation of a grayscale image and the following list of configurable parameters:

- Gaussian Blur (σ)
- Gaussian Kernel Size
- Derivative Kernel Type (Sobel, Prewitt, Scharr)
- Neighbor Depth (number of neighbors in each direction for NMS)
- Pixel Connectivity (4-connected vs 8-connected for NMS)

- High threshold (0.0-1.0)
- Low threshold (0.0-1.0)
- Max Hysteresis Iterations

Each stage of the Canny edge detection algorithm was implemented as an independent class method and returned the processed image data from that stage. This allowed a particular stage in the algorithm to be executed and examined. If the data from the previous stage was unavailable, the current stage would execute the previous stage. Each stage performed this action until all preliminary data was processed and available.

The class also featured "getter" methods to retrieve the processed image data from any of the stages after processing was complete. "Setter" methods were also provided to allow adjustment of an object's original image and parameters. Both types of methods were important for visual and quantitative analysis of the canny-ecen898 performance.

3.2 Visual Evaluation

The initial stage of the experiment was conducting visual comparisons of the original image with the edge image generated by canny-ecen898. This served two purposes: fine tuning the canny-ecen898 implementation and exploring the effects of the configurable parameters.

A simple "playground" script was used in the fine tuning stage. This script executed the OpenCV Canny edge detector and used OpenCV to display the canny-ecen898- and the OpenCV-generated edges side-by-side. This revealed several issues in the implementation such as an indexing error when comparing diagonal neighbors in the non-maximum suppression implementation. It also helped determine the computationally inefficient stages of the Canny implementation. The playground script was sufficient for this purpose because frequent manipulation of parameters was not necessary.

More iterations were required for exploring the effects of the canny-ecen898 parameters. Here, images were selected based on characteristics that would evaluate canny-ecen898 performance given different edges. The images used for this exploration are provided in Table 1 with a brief description of why they were chosen.

Table 1: Images used for visual evaluation of canny-ecen898 with selection criteria.

Image	**Selection Criteria**
lena.bmp	Provides a good combination of many small, random direction edges in the feather and large, curved edges in the face, hat, and background objects. Presents a challenge to detect the feather edges without also detecting too much noise.
cameraman.tif	Considerable number of strong edges in the foreground with little intensity variations. Additional strong edges in the background with noisy background presented by the grass. The prominent foreground and background edges provide opportunity to explore the state when background element recede and how much of the foreground is maintained in that case.
braves.jpeg	Provides human subjects occluded by the presence of confetti in the foreground. Presents a challenge to try and retain the human edges while trying to eliminate confetti edge.

The parameters were updated iteratively to visually compare the generated edges at each step. For example, the high threshold was manipulated until performance was maximized or until some goal was achieved (e.g. background edged disappeared). The low threshold was manipulated next to see if the edges could be further improved. Then the Gaussian blur and kernel size were adjusted. Such step-wise modifications of single parameters allowed the effects of a particular parameter to be evaluated.

The iterative nature required constant changes and recalculation of edges and the playground script mentioned previously was too cumbersome. The browser-based graphical user interface (GUI) shown in Figure 4 was developed to aid the exploration effort.

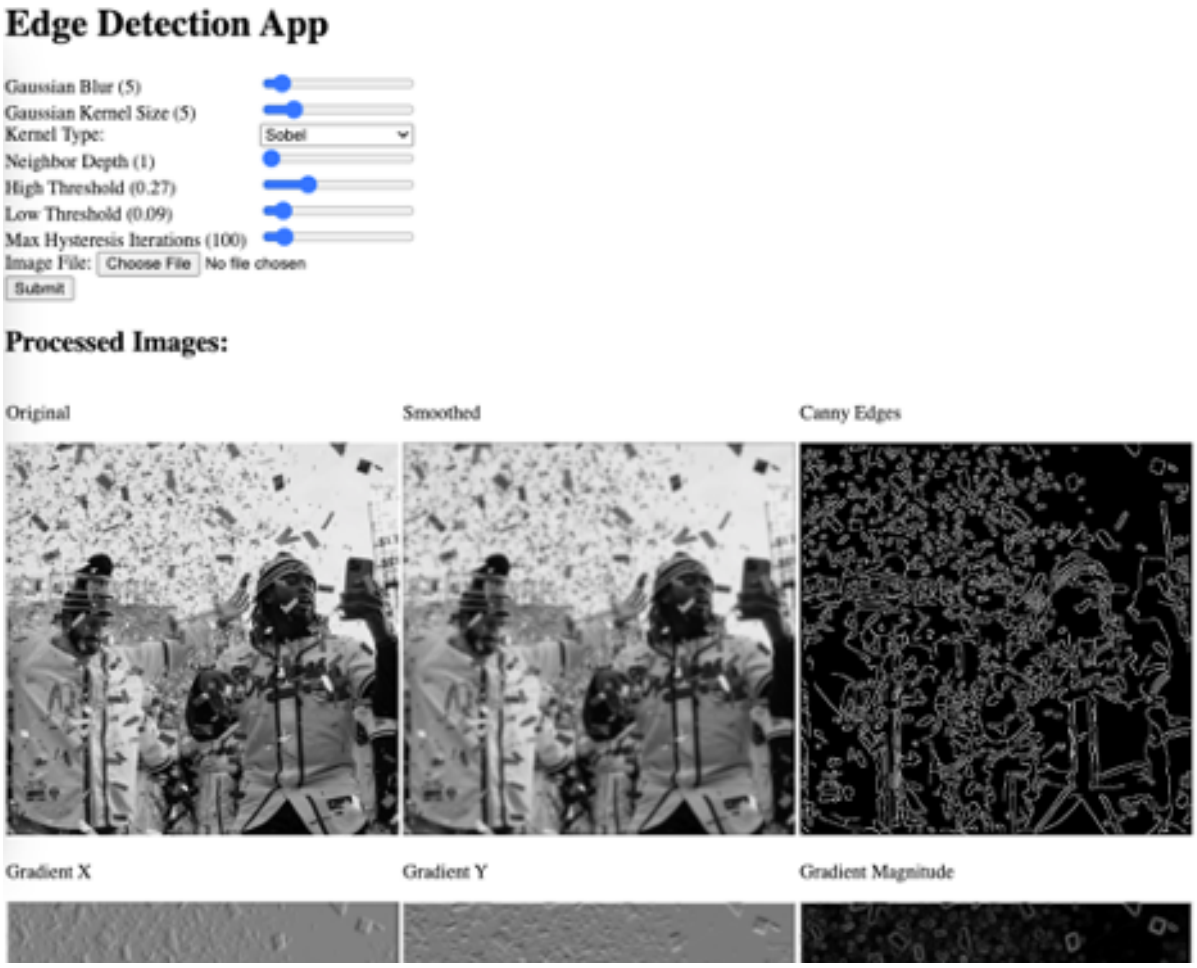

Figure 4: Browser-based tool for rapidly manipulating Canny edge detector parameters and viewing the output of each stage

The GUI was developed using the Flask Python library that provides API for creating micro-web services. Running the application stands up a small HTTP server that serves a HTML page that presents the user-input form and the processed images. This page may be accessed from any browser. The server handles the posted form on submission, creates and canny-ecen898 object with the supplied parameters, performs edge detection, and returns a 3x3 array of images to the browser. The nine returned images were the processed image arrays output from each stage of the Canny algorithm.

3.3 Quantitative Evaluation

Quantitative evaluation requires ground truth data for comparison of the edge detector's predicted edges. Many of the publications listed in [4] utilized the Berkeley Segmentation Dataset 300 (BSDS300), which provides images and human annotated edges. However, the human annotations for BSDS300 are primarily focused on larger segmentation boundaries. As such, there are many edges in the image that are not accounted for in the ground truth data and BSDS300 proved to be a poor dataset for this experiment. The review in [4] listed another publication [5] that utilized simulated data for ground truth. The simulated dataset in [5] could not be located; however, the approach inspired development of image simulation script. This script generated a binary edge image of shapes, a filled-shape grayscale image, and two noisy images with small ($\sigma = 5$) and large ($\sigma = 20$) Gaussian white-noise applied as shown in Figure 5.

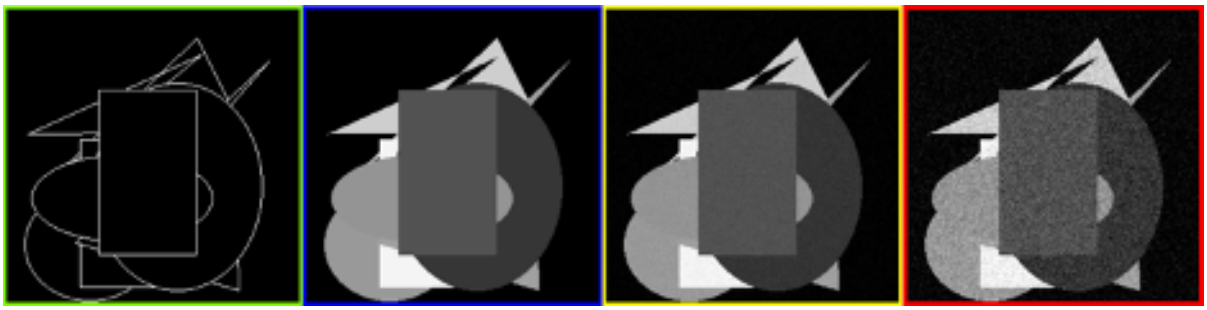

Figure 5: Simulated images for quantitative evaluation of canny-ecen898. Left-to-right: Ground truth edges, filled edges, $\sigma = 5$ noisy image, $\sigma = 20$ noisy image.

The script generated randomly located ellipses, rectangles, and randomly generated polygons with a one-pixel width white border and black fill. The black fill ensured that occluded edges in the filled shapes would not be included in the ground truth edges. The various shapes ensured that edge detector would be tested on different edge orientations and sizes. Furthermore, noise was applied to better simulate the noise present in real digital images.

The true positive rate (TPR) and false positive rate (FPR) were discussed in 2.5 as two good metrics for evaluating the detectors ability to correctly classify edges and non-edges. If TPR is plotted as a function of FPR, a receiver operating characteristic (ROC) curve is generated. The diagonal from (0,0) to (1,1) represents a random classifier, i.e. a classifier that simply guesses. Therefore, a good classifier or edge detector in this case would drive the curve towards the upper-left corner of the plot, increasing the area under the curve (AUC).

Another script was developed that automated the discovery of optimal parameters to maximize the AUC. A set of 50 simulated images similar to the one shown in Figure 5 were divided into a training set (40) and a test set (10). The binary image was used as ground truth and the noisy image was used for edge detection. The goal was that using the noisy image would generalize to the less noisy images also. All combinations of the following parameters were created:

- Gaussian Blur (σ): 3, 8, 21, 34
- Gaussian Kernel Size: 3, 5, 9, 13
- Derivative Kernel: Sobel
- Neighbor Depth: 1, 2, 3

- Pixel Connectivity: 8-connected
- High threshold: 0.15, 0.27, 0.39, 0.51, 0.67
- Low threshold: 0.05, 0.09, 0.13
- Max Hysteresis Iterations: 100

These parameters are not exhaustive but they provide a strong basis to build on for later research. This resulted in 720 combinations. The edge detection was performed on the 40 training images and the AUC was calculated from the 40 generated TPR and FPR scores. This was repeated for each combination for a total of 28,800 edge detections. At each step, if the AUC was higher than the previous maximum, the maximum was updated and the associated parameter combination was saved. After all iterations, the maximum AUC score and associated parameters were presented. The optimal parameters were applied to the test set to determine how well they generalized.

4 Results

4.1 Computational Performance

The playground script was useful for comparing the computational efficiency of canny-ecen898 to the OpenCV Canny implementation. Table 2 shows the execution times for various stages of the Canny algorithm. The canny-ecen898 implementation takes nearly 55x longer than the OpenCV implementation. Much of this is caused by canny-ecen898's operations that require iterating over matrices (i.e. smoothing, gradient, NMS, and hysteresis. However, operations like gradient magnitude calculation could utilize NumPy's built-in operations that accelerate matrix calculations. The slowness of canny-ecen898 can most likely be attributed to the slowness of Python looping operations. Attempts were unsuccessful to apply NumPy's broadcasting features to increase the speed of these operations and SciPy and OpenCV library functions were avoided for the purposes of educational implementation. Additionally, the slowness was a key factor in deciding the limited data size for acquiring quantitative results.

Table 2: Execution times of canny-ecen898 compared to OpenCV equivalent (if possible) for detecting edges of cameraman.tif.

	Total	**Smoothing**	**Gradient**	**Magnitude**	**NMS**	**Hysteresis**
canny-ecen898	1.7 s	342 ms	689 ms	3.8 ms	143.5 ms	545.4 ms
opencv	31 ms	5.7 ms	2.5 ms	3.5 ms		

4.2 Qualitative Results

The primary objective of the experiment was to implement a functional Canny edge detector in Python. Figure 6 provides a sequence of processed data from the original image to the final demonstrating that the canny-ecen898 implementation is a functional Canny edge detector. Blurring is observed in the top, second-from left frame. The emphasis of vertical and horizontal lines are evident for the x- and y- gradients in the next two frames, respectively. The gradient magnitude in the top, far-right frame shows a better representation of the horizontal, vertical, and diagonal lines. The non-maximum suppressed image in the bottom, far-left frame appears

to have reduced the thickness of the gradient magnitude edges. The weak edges in the next frame appear to consist of the noisier, smaller edges while the strong edges in the following frame appear to be compromised of larger edges primarily in the foreground. Lastly, the hysteresis operation is demonstrated correctly by presence of edges from the weak edge image that were not present in the strong edge image, most notably the tower in the background.

Figure 6: Sequence of images generated by canny-ecen898 on cameraman.tif at each stage of the algorithm. Left to right: original, smoothed, x-gradient, y-gradient, gradient magnitude, non-max suppression, weak edges, strong edges, and final Canny detected edges.

The secondary objective was to explore the effects of various parameters on the detected edges. The parameters could be grouped by subtle-effect parameters and major-effect parameters. The subtle-effect parameters include derivative kernel type, pixel connectivity, and neighbor depth. Increasing the neighbor depth for NMS decreased the amount of closely parallel lines in the final edge detection as shown in Figure 7. This makes sense because more parallel edges would get suppressed when more edges are compared for NMS.

Figure 7: Increasing neighbor depth from one pixel (left) to four pixels (right) decreases the amount of parallel lines.

Other subtle effects include the amount of noisy, finer details increased and gaps in prominent edges decreased in order of Prewitt, Sobel, and Scharr 3x3 derivative kernels. This makes sense as the derivative estimation is generally considered to improve in that order. For pixel-connectivity, additional less-prominent diagonal edges were detected when going from four-connected to eight connected. Again, this makes sense because provides eight-connected pixels allows for analysis of local maxima in the diagonal directions as well as horizontally and vertically.

The major-effect parameters include the amount of Gaussian blur (σ), the size of the Gaussian kernel, the lower dual threshold, and the upper dual threshold. Their effects are demonstrated on cameraman.tif, lena.bmp, and braves.jpeg in Figure 8.

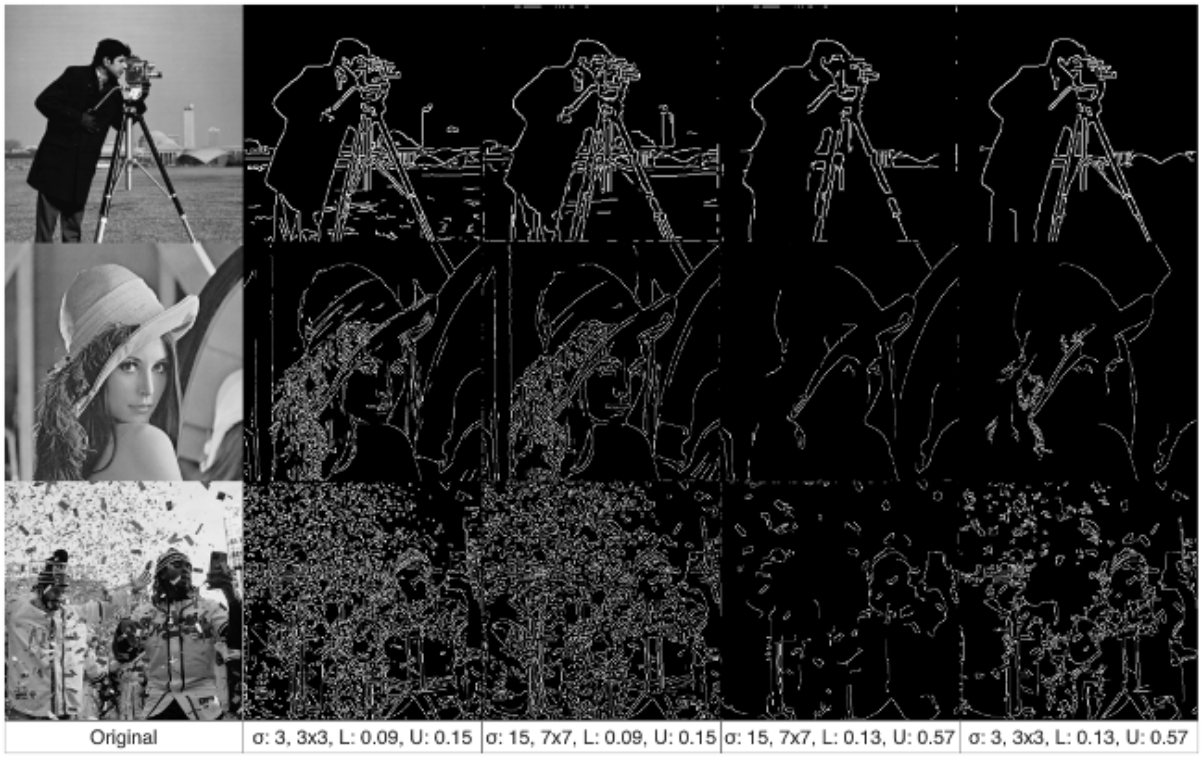

Figure 8: Comparison of the effects of Guassian blur and dual threshold values.

At low blur and low thresholds (second column from left), finer details appear in abundance. For "cameraman" (top) and "lena" (middle) this may be considered good because there are fine details in the camera and feather, respectively, that may be of interest. However, for "braves" (bottom) this is not ideal due to the presence of all the small confetti details that make it difficult to recognize the people in image. Moving right, the blur is increased to a large value. Some of the fine details disappear like the grass in "cameraman" and eye details in "lena"; however, at this high of blur some distortions occur to the edges of "cameraman". The smaller confetti begin to suppress in "braves" but it is still difficult to identify the people. At high blur and high thresholds (second column from right), much of the fine detail is gone and only the prominent identifying edges are visible. Some foreground edges of "cameraman" disappear like part of the leg and camera leg details and there is no evidence the "lena" has a feather in the hat. However, it is very clear that "cameraman" features a person using a device on a tripod and "lena" is clearly a person wearing a hat. Furthermore, one person is very evident

in "braves" with most of the confetti eliminated. Lastly, at low blur and high thresholds (far right column), some edges return but it's still primarily the prominent foreground edges. The feather is somewhat distinguishable in "lena". More confetti is present in "braves" but the second person is also more identifiable. These images and the comparison of detected edges demonstrate the power of configurable parameters to achieve various goals depending on the edge detection application.

Another observation was the limiting effect that the size of the Gaussian kernel had on the blurring effect of larger σ values. This makes sense because a larger σ means a wider Gaussian but the effect of that width is limited if there aren't more pixels in the kernels to distribute across. Larger σ saturate the kernel at some point.

4.3 Quantitative Results

The use of simulated image data to provide ground truth for pseudo-optimizing the canny-ecen898 parameters was able to achieve an AUC of 0.7317 on the training set. The parameters that maximized the AUC were:

- Gaussian Blur (σ): 3
- Gaussian Kernel Size: 3
- Derivative Kernel: Sobel
- Neighbor Depth: 1
- Pixel Connectivity: 8-connected
- High threshold: 0.15
- Low threshold: 0.09
- Max Hysteresis Iterations: 100

These parameters generalized well to the test, achieving an AUC of 0.7131. These parameters were applied to an example image as shown in Figure 9. The parameters appear to provide a good balance between prominent edges and fine details. For example, the body shapes and facial features are clearly expressed while the fine details dress pattern and the reflected tree branches are preserved. However, other details are suppressed like the majority of wrinkles and creases in the clothing.

These "optimized" parameters provide a good generalization but not a complete picture. They assume the goal is to preserve as many edges from the image while minimizing false edges. However, this goal is not beneficial if a user wants to suppress the less prominent background edges like the reflected branches in Figure 9. This goal assumes that some true edges are irrelevant and the AUC metric would be driven down by this assumption. Hence, the optimal parameters for eliminating background noise would not be discovered with the current experimental setup. It may be possible to simulate images with noisy background edges and perform an experiment where these edges are masked in the ground truth image, but there will be many other situations that warrant specific needs. In other words, edge detection becomes complex when considering multiple different use cases. This is why a configurable edge detection implementation may be considered best because it provides a generic tool that can be configured to suit the needs of a specific application.

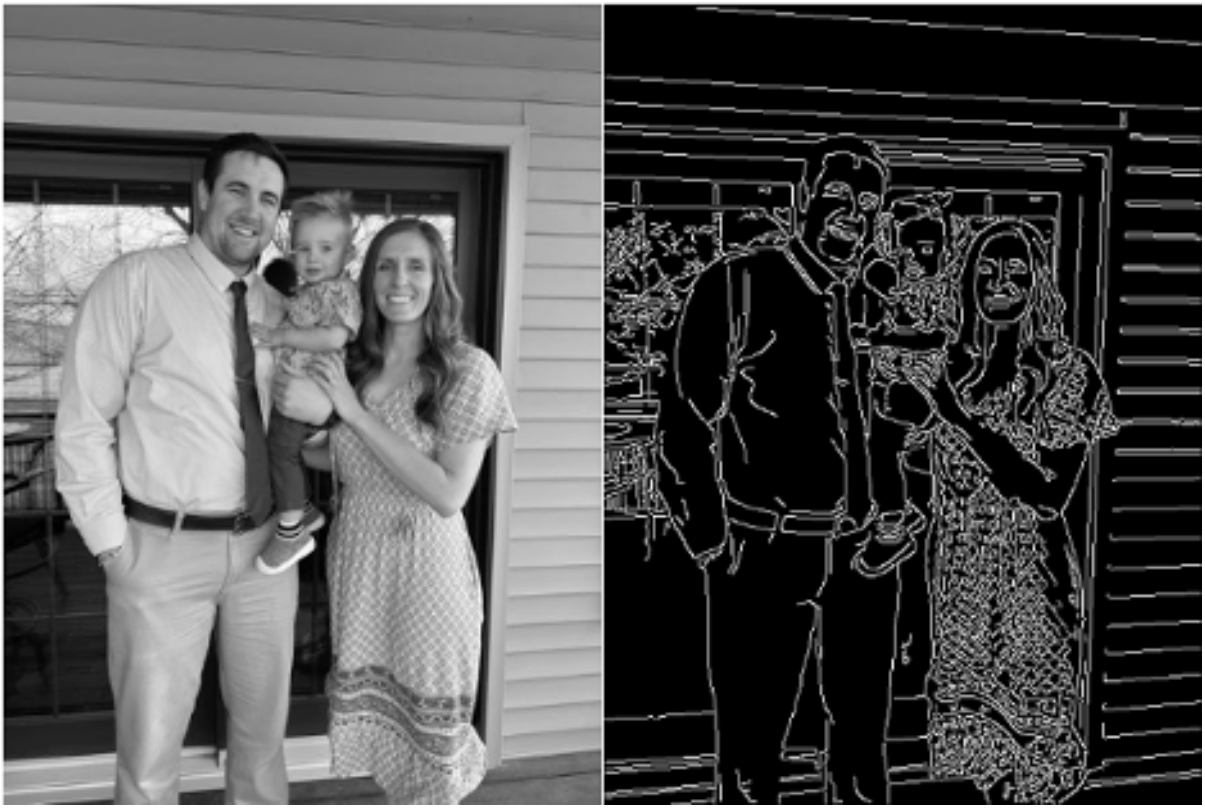

Figure 9: Example edge detection using the pseudo-optimized parameters.

5 Conclusion

The Canny edge detector is a significant and popular edge detection algorithm. Therefore, it is an important concept for computer vision students to explore through practical implementation. Therefore, the goals of this experiments were to implement a functional Canny edge detector in a modern programming language (Python) and explore the effects of various parameters of the algorithm to the final detected edges.

The Canny edge detection algorithm was presented as a multi-part algorithm with each part briefly discussed. This includes image smoothing, gradient estimations for edge detection, non-maximum suppression, dual thesholding, and hysteresis. Each part was successfully implemented within an overarching CannyEdgeDetector Python class. The successful implementation was demonstrated through visual analysis of the data output from each stage; hence, the primary goal was achieved.

Additional tools were developed in Python to perform visual and quantitative analysis of the detected edges in response to adjusting several parameter. This resulted in a grouping of the parameters into subtle-effect and major-effect parameters. The subtle-effect parameters included NMS neighbor depth, NMS pixel connectivity, and derivative kernel type. The major-effect parameters included amount of Gaussian blur (σ), Gaussian kernel size, and the upper and lower thresholds for dual thresholding. Additionally, pseudo-optimized parameters were identified that maximized the amount of true edges and minimized the amount of false edges. Hence, the second goal of exploring the effects of algorithm parameters was achieved. Furthermore, the optimized parameters were a partial-truth because they only maximized for

a particular use-case. This provided a key takeaway that a configurable edge detection implementation may be considered best because it provides a generic tool that can be configured to suit the needs of a specific application.

References

[1] W. E. Snyder and H. Qi, *Fundamentals of Computer Vision*. Cambridge University Press, 2017.

[2] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986. DOI: 10.1109/TPAMI.1986.4767851.

[3] OpenCV. "Opencv canny edge detection python tutorial." Accessed: February 29, 2024. (2024), [Online]. Available: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html.

[4] N. Tariq, R. Hamzah, T. F. Ng, S. Wang, and H. Ibrahim, "Quality assessment methods to evaluate the performance of edge detection algorithms for digital image: A systematic literature review," *IEEE Access*, vol. PP, pp. 1–1, 2021. DOI: 10.1109/ACCESS.2021.3089210.

[5] S. Alpert, M. Galun, B. Nadler, and R. Basri, "Detecting faint curved edges in noisy images," Sep. 2010, pp. 750–763, ISBN: 978-3-642-15560-4. DOI: 10.1007/978-3-642-15561-1_54.