Edge Detection in Digital Image Processing

Ruixiao Duan

Abstract

This paper illustrates the fundamental concepts of various edge detectors and how to apply them to find edges in image processing. The main two kinds of operators are Gradient and Laplacian operators. Besides, Canny edge detector also has perfect performance when processing noisy images and is supposed to give thin, bright edge. This paper describes how these edge detectors work, and displays performances of software implementations in MATLAB. Finally, the paper compares the differences between multiple types of detectors and gives some explanations.

I. Introduction

EDGE detection is a very important area in the field of computer vision. Edges in images are areas with strong intensity contrasts - a jump in intensity from one pixel to the next. Edges define the boundaries between regions in an image, which helps with segmentation and object recognition. They can show where shadows fall in an image or any other distinct change in the intensity of an image. Edge detection is a fundamental of low-level image processing and good edges are necessary for higher level processing [1]. Edge detection can describe the target outline, the relative position within the target area, and other important process in image processing.

The problem is that in general edge detectors behave very poorly. While their behavior may fall within tolerances in specific situations, in general edge detectors have difficulty adapting to different situations. The quality of edge detection is highly dependent on lighting conditions, the presence of objects of similar intensities, density of edges in the scene and noise. While each of these problems can be handled by adjusting certain values in the edge detector and changing the threshold value for what is considered an edge, no good method has been determined for automatically setting these values, so they must be manually changed by an operator each time the detector is run with a different set of data.

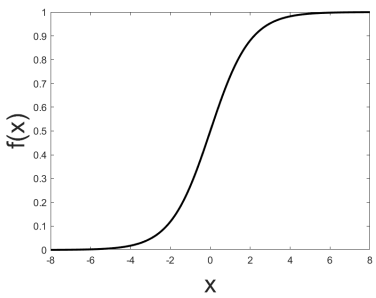
Since different edge detectors work better under different conditions, we would like to see performances of different operators in different cases. We tested several edge detectors that use different methods for detecting edges and compared their results under a variety of situations to determine which detector was preferable under different sets of conditions.

Now list what we finished in this project:

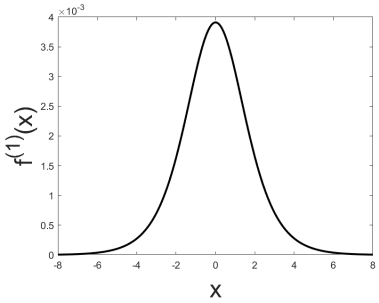
- Studied edge detection algorithms: gradient, Laplacian and Canny detection algorithms.
- Implemented Sobel, Prewitt, Roberts, Laplacian of Gaussian (LoG) and Canny edge detectors [1] in MATLAB.
- Trained dataset and compared with true labels to find proper hyperparameters.
- Tested detectors with test dataset and evaluated their performance.
- Made comparison with different detectors and try to explain reasons.

II. Related work

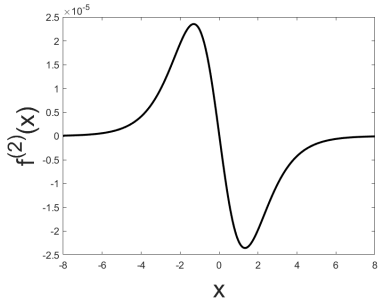
Assume we have a 1-dimensional signal $f(x)$ displayed in Fig. 1a, it is easy to find there is a sharp change at $x = 0$ from the image. However, how can people get this result mathematically? Obviously, if there is a large change at x_0, then the first derivative of $f(x)$ will have a large value at x_0, meaning $f^{(1)}(x_0)$ is a local maximum, which is shown in Fig. 1b. Moreover, the second derivative of $f(x)$ will be zero at x_0, shown in Fig. 1c with a peek and a valley near x_0. But, the second-derivative method is more sensitive to noise effect than first-derivative one. Besides, higher order derivatives also works but they are much more possible influenced by noise and computation is also a huge work. Thus, people seldom applied higher order derivative to find sharp change.



(a) Original signal has a sharp increase at $x = 0$. The sharp increase can be viewed as an edge.



(b) First derivative of original signal, a peek is detected at $x = 0$. Usually this peak is positive or negative.



(c) Second derivative of original signal, there is a positive and negative peek before, after original point.

Fig. 1: A mathematical representation of sharp increment in 1-dimensional signal.

Edge is defined by area with strong intensity contrasts. Since horizontal and vertical direction are orthogonal, if a pixel in images has sharp variety along any direction, then there may be an edge at this pixel. Considered the discussion about 1-dimensional signal, it is similar to find solutions for images.

Generally, there are two different types of operators used in image processing for edge detection: Gradient and Laplacian operators [2]. For gradient based operators, which looks for the maximum value of the first derivative, Roberts, Sobel and Perwitt edge detectors are widely used, and Laplacian based edge detectors, which tries to find zero-crossing of the second derivative, Laplace and LoG detectors, and besides, Canny edge detector [3], which gives a thin and bright edges, are also common.

A. Gradient Operators

The gradient operator (Sobel, Perwitt, Roberts) performs a 2-dimensional spatial gradient measurement on images [4]. It uses a pair of horizontal and vertical gradient matrices, usually whose dimension are 3×3, to find the maximum value of the first derivative [5].

Generally, the expression can be written as

$$\nabla I[m,n] = \left[\frac{\partial I[m,n]}{\partial m}, \frac{\partial I[m,n]}{\partial n}\right], \tag{1}$$

which is the first differential of image $I[m,n]$ along two directions. For different operators, the differential matrices are different. It will be illustrated in section *Gradient Detection Algorithms Introduction*.

Usually the dimension of masks is 3, however, in recent application, it is helpful to introduce a higher dimensional matrix for masking [6]. The operator will give thinner and brighter edges as parameters are set properly.

B. Laplacian Operators

The Laplacian operator (Laplace, LoG) works in a similar way as gradient operator. The only difference is that, instead of first differential, Laplacian operator finds edges based on zero-crossing of the second differential. The expression can be generally written as

$$\nabla^2 I[m,n] = \left[\frac{\partial^2 I[m,n]}{\partial m^2}, \frac{\partial^2 I[m,n]}{\partial n^2}\right], \tag{2}$$

which is the second differential of image $I[m,n]$ along two directions. Details will be introduced in the following section *Laplacian Detection Algorithms Introduction*.

C. Canny Edge Detection

Although the representative first order differential operators (Roberts, Prewitt, Sobel) and second order differential operators (Laplace, LoG) have many advantages such as simple computation, rapid speed and easy to implement, they are more sensitive to noise and their detection effect are not perfect in engineering application [7].

In 1986, Canny proposed three criteria to judge edge detection operator performance: SNR criterion, localization precision criterion and single edge response criterion, and deduced the best Canny edge detection operator [8]. Compared with common edge detection algorithm, in most cases, the Canny algorithm has the best performance [9] [10]. In recent years, some researchers offered many improved algorithms based on Canny algorithm and applied them to practical engineering. Er-Sen Li improved the image gradient magnitude calculation operator and automaticity of edge detection by Otsu's threshold selection method, and it showed good edge detection results to some extent [11]. Again S. introduced an improved Canny operator for Asphalt Concrete (AC) applications [12]. Xiangdan Hou proposed an improved Canny algorithm based on the Histogram-based fuzzy C-means clustering algorithm. It was applied in detecting road surface distress image and it appeared good effect [13].

Since the image quality will be influenced by some factors such as noise and illumination in the process of the image acquisition, whats more, to the large view scope of images, its local contrast is different from each other, the parameters of the traditional Canny edge detection algorithm are fixed and cannot adapt to the edge detection process in different conditions. Therefore, this paper improved the image gradient calculation operator, which is helpful to preserve more useful detail edges and more robust to noise. Two adaptive threshold selection methods were presented for two kinds of typical images respectively, and it can contribute to fit diffident conditions automatically.

III. Project Perspectives

A. Implementation

In this project, we are supposed to study how these detectors work and implement them in MATLAB. There exists a function, *edge.m*, which is in digital image processing toolbox and outputs a binary image after edge detectors. During implementation, we are going to refer some details from this built-in function. Besides, we will conclude performances between different types of detectors and try to give some reasonable explanations.

B. Evaluation

The goal of the benchmark is to produce a score for an algorithm's boundaries for two reasons: (1) So that different algorithms can be compared to each other, and (2) So that progress toward human-level performance can be tracked over time. We search some papers online about how to benchmark, finally decide to follow a criterion below.

The dataset we will use in this project is an online dataset called the Berkeley Segmentation Dataset and Benchmark (BSDS500) [14]. The dataset consists of lots of natural images with ground-truth human annotations.

The setup of benchmark is as follows. The human segmented images provide our ground truth boundaries. We consider any boundary marked by a human subject to be valid. Since we have multiple segmentations of each image by different subjects, it is the collection of these human-marked boundaries that constitutes the ground truth. We are then presented the output of some algorithm for an image. Let us assume that this output is a soft boundary map with one pixel wide boundaries, valued from zero to one where high values signify greater confidence in the existence of a boundary. Our task is to determine how well this soft boundary map approximates the ground truth boundaries.

Traditionally, one would "binarize" the boundary map by choosing some threshold. There are two problems with thresholding a boundary map: (1) The optimal threshold depends on the application, and

we would like the benchmark to be useful across different applications, and (2) Thresholding a low-level feature like boundaries is likely to be a bad idea for most applications, since it destroys much information. For these reasons, our benchmark operates on a non-thresholded boundary map.

Nevertheless, we do need to threshold the boundary map in order to compare it to the ground truth boundaries, but we do so at many levels, e.g. 30. At each level, we compute two quantities – **precision** and **recall** – and in this manner produce a precision-recall curve for the algorithm. Precision and recall are similar to but different from the axes of ROC curves. Precision is the probability that a machine-generated boundary pixel is a true boundary pixel [14]. Recall is the probability that a true boundary pixel is detected [14]. We consider these axes to be sensible and intuitive. Precision is a measure of how much noise is in the output of the detector. Recall is a measure of how much of the ground truth is detected. The curve shows the inherent trade-off between these two quantities – the trade-off between misses and false positives – as the detector threshold changes.

Although the precision-recall curve for an algorithm is a rich descriptor of its performance, it is still desirable to distill the performance of an algorithm into a single number. This is possible to do in a meaningful way for algorithms whose curves do not intersect and are roughly parallel. When two precision-recall curves do not intersect, then the curve furthest from the origin dominates the other. The summary statistic that we use is a measure of this distance. It is the F**-measure**, which is the harmonic mean of precision and recall given as follow [14],

$$F = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}. \tag{3}$$

The F-measure is defined at all points on the precision-recall curve. We report the maximum F-measure value across an algorithm's precision-recall curve as its summary statistic.

Why do we use precision-recall curves instead of ROC curves?

Receiver operating characteristic (ROC) curves show, qualitatively, the same trade-off between misses and false positives that precision-recall curves show. However, ROC curves are not appropriate for quantifying boundary detection. The axes for an ROC curve are fallout and recall. Recall is the same as above, and is also called hit rate. Fallout, or false alarm rate, is the probability that a true negative was labeled a false positive. This is not a meaningful quantity for a boundary detector since it is not independent of the image resolution. If we reduce the radius of the pixels by a factor of n so that the number of pixels grows as n^2, then the number of true negative will grow quadratically in n while the number of true positives will grow only linearly in n. Since boundaries are 1D objects, the number of false positives is most likely to also grow linearly in n, and so the fallout will decline by a factor of $1/n$. Precision does not have this problem, since instead of being normalized by the number of true negatives, it is normalized by the number of positives.

IV. Gradient-Based Detection Methods

In this section, we are going to discuss the basic idea about gradient detectors. Generally there are three steps: we apply different operators to filter image to get x and y direction components. Then get its gradient magnitude image. Finally, binarize the gradient magnitude image to get a binary edge image.

A. Operators (Masks)

There are three different operators or masks, Sobel, Prewitt and Roberts widely applied, which are given as follow. After these masking, the increment or decrement in x and y direction will be saved in two images.

1) Sobel:

$$P_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, P_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}. \tag{4}$$

2) Prewitt:

$$P_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}, P_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix}. \tag{5}$$

3) Roberts:

$$P_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, P_y = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}. \tag{6}$$

B. Gradient Magnitude

Depend on the operators in x and y directions, we can get the gradient magnitude,

$$\begin{aligned} G_x[m,n] &= P_x * I[m,n], \\ G_y[m,n] &= P_y * I[m,n]. \end{aligned} \tag{7}$$

Then combine these two components to get the magnitude of gradients,

$$E_m[m,n] = \sqrt{G_x[m,n]^2 + G_y[m,n]^2}. \tag{8}$$

C. Image Binarization

Apply a threshold to binarize gradient image to an edge image $BW[m,n]$ so that we can finally get a clear image with edges. For any pixel with value above threshold, label it as edge; otherwise, label as background.

V. Laplacian-Based Detection Methods

We have already described that Laplacian detection algorithm is trying to find zero-crossing point between a hill and a valley shown in Fig. 1c. However, in calculating second derivative is very sensitive to noise. This noise should be filtered out before edge detection. To achieve this, LoG is used. This method combines Gaussian filtering with the Laplacian for edge detection.

A. Laplacian of Gaussian (LoG)

In LoG edge detection uses three steps in its process. The first one is filter image object. Secondly, it enhances the image object and finally detects. Here, Gaussian filter is used for smoothing and the second derivative is used for the enhancement step. In this approach, at first the noise is reduced by convoluting the image with a Gaussian filter. The isolated noise points and small structures are filtered out. However the edges are spread with smoothing. Those pixels are locally maximum gradient which are considered as edges by the edge detector in which the zero crossings of the second derivative are used. The zero crossings are only insignificant edges to avoid the detection that corresponds as the first derivative is above some thresholds, which are selected as edge points. The edge direction is obtained using the direction in which zero crossing occurs. Concluded, the LOG mask can be expressed as

$$LoG[m,n] = -\frac{1}{\pi\sigma^4}\left[1 - \frac{(m-k)^2 + (n-k)^2}{2\sigma^2}\right] exp\left\{-\frac{(m-k)^2 + (n-k)^2}{2\sigma^2}\right\}, \tag{9}$$

where m and n are coordinates of image and $0 \leq m, n \leq 2k$, $2k+1$ is the length of filter window, and σ is the variance of Gaussian distribution.

So the filtered image is given by

$$J[m,n] = I[m,n] * LoG[m,n]. \tag{10}$$

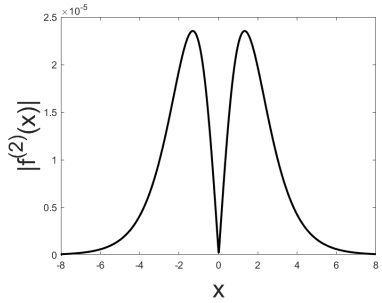


Fig. 2: Absolute value of second derivative.

However, after we apply second derivative computation, it is not reasonable to say that all zero-crossing points are edge points. The LoG operator calculates the second spatial derivative of an image. This means that in areas where the image has a constant intensity (i.e. where the intensity gradient is zero), the LoG response will be zero. In the vicinity of a change in intensity, however, the LoG response will be positive on the darker side, and negative on the lighter side like Fig. 2. This means that at a reasonably sharp edge between two regions of uniform but different intensities, the LoG response will be:

- zero at a long distance from the edge,
- positive just to one side of the edge,
- negative just to the other side of the edge,
- zero at some point in between, on the edge itself.

Thus, we get the absolute value of filtered image. Now the edge is represented by a dark line between two bright lines. When viewing the result, the edges look like smooth and doubled. Now the image becomes

$$J_{abs}[m,n] = \text{abs}(J[m,n]). \tag{11}$$

Finally, in order to obtain a direct, sharp edge image $BW[m,n]$, we apply image binarization by a threshold. If the pixel value is above the threshold, set the pixel value to 1; otherwise, set it to 0.

VI. Canny Edge Detection Method

Besides gradient and Laplacian detection algorithm, we would like to illustrate Canny detection algorithm. There are several steps for this algorithm. We will explain it step by step.

A. Gaussian Smooth

Apply a Gaussian filter to smooth image and remove noise. Assume $I[m,n]$ is the original image given, and $J[m,n]$ is the filtered image. Then we have

$$J[m,n] = I[m,n] * G[m,n], \tag{12}$$

where G is a zero mean Gaussian filter with std σ. The transfer function is given by

$$G[m,n] = \frac{1}{2\pi\sigma^2} exp\{-\frac{(m-k)^2+(n-k)^2}{2\sigma^2}\}, \tag{13}$$

where $0 \leq m,n \leq 2k$, and $2k+1$ is the length of Gaussian window.

B. Intensity Gradient

For each pixel, compute the image gradient:

$$\bigtriangledown J[m,n] = (J_m[m,n], J_n[m,n]). \tag{14}$$

We can apply either Prewitt and Sobel to find gradients. Two 3×3 kernels is to get the vertical and horizontal derivative approximations.

For Sobel operator, we have

$$P_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, P_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}. \tag{15}$$

For Prewitt operator, we have

$$P_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}, P_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix}. \tag{16}$$

Thus we can get the gradients by

$$\begin{aligned} J_m[m,n] &= P_x * J[m,n], \\ J_n[m,n] &= P_y * J[m,n]. \end{aligned} \tag{17}$$

Then, from the two directions' gradient, we can get the intensity estimation $E_s[m,n]$ and direction estimation $E_o[m,n]$ by

$$\begin{aligned} E_s[m,n] &= \sqrt{J_m[m,n]^2 + J_n[m,n]^2}, \\ E_o[m,n] &= \arctan\left(\frac{J_m[m,n]}{J_n[m,n]}\right). \end{aligned} \tag{18}$$

Since detected edges are in directions in 360°, it is hard to determine the direction of one edge. We would like to generalize directions in four directions $\{-\pi/2, -\pi/4, 0, \pi/4\}$.

This step can be concluded as

$$E_o[m,n] = \begin{cases} -\pi/2, & \text{If } 3\pi/8 \le E_o[m,n] < -3\pi/8. \\ -\pi/4, & \text{If } -3\pi/8 \le E_o[m,n] < -\pi/8. \\ 0, & \text{If } -\pi/8 \le E_o[m,n] < \pi/8. \\ \pi/4, & \text{If } \pi/8 \le E_o[m,n] < 3\pi/8. \end{cases} \tag{19}$$

C. Non-Maximum Suppression

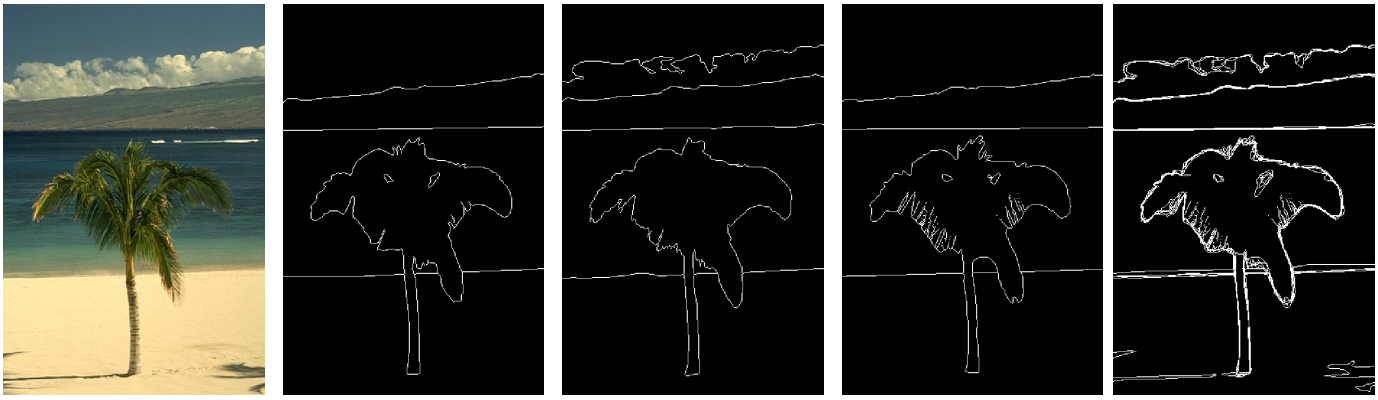
Apply non-maximum suppression to get rid of spurious response to edge detection.

Define $I_N[m,n]$ is the thinned edge image. If $E_s[m,n]$ is smaller than at least one of its neighbor along d, then $I_N[m,n] = 0$; otherwise, $I_N[m,n] = E_s[m,n]$.

D. Double Threshold

Apply double threshold to determine potential edges. Define low threshold as L and high threshold as H, then this step can be concluded as:

$H \le I_N[m,n]$	strong edge
$L \le I_N[m,n] < H$	weak edge
$I_N[m,n] < L$	not an edge



(a) Original. (b) Subject1. (c) Subject2. (d) Subject3. (e) Combined.

Fig. 3: This figure shows three different boundaries by different subject. 3a is colorful and the rest boundary images are binary which white color represents edges. We can find there are some slight differences between these subject true boundaries. Thus, when we use these labels, we treat any pixel labeled by any subject as an edge pixel. 3e shows the combined result. It is easy to find edges in combined image are much wider and brighter than subject edge images.

E. Track edge by hysteresis

For each weak edge pixel, if there is a strong edge pixel in its 8-connected neighbor pixel, then it changes to a strong edge; otherwise, it is no edge.

Then we label all strong pixel as pixel, and the rest as background. After this step, we can finally get a binary edge image $BW[m, n]$.

VII. Implementation

A. Dataset Overview

The dataset is available online. The dataset and source codes are accessible from Github[1].

In order to evaluate our edge detection algorithms, we are going to use a dataset we mentioned before. The dataset consists of 500 natural images, ground-truth human annotations. The data is explicitly separated into disjoint train and test subsets. The images are divided into a training set of 300 images, and a test set of 200 images.

For each image, several people draw its boundary manually. Thus, we have multiple true boundary images for one original image like Fig. 3.

B. Dataset Preprocessing

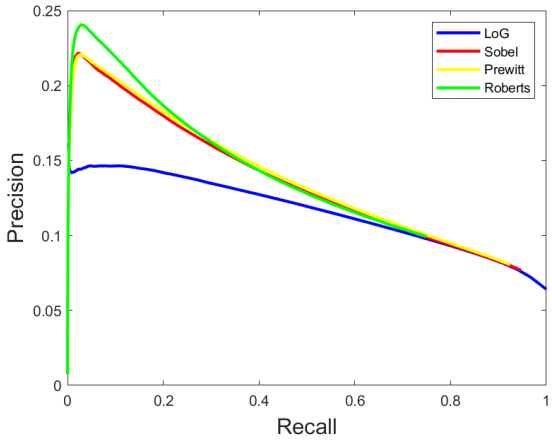
As mentioned before, for any original image, there are several subject true boundary images by different people. The first thing before processing is that we need to combine these true boundary images to one image. Still displaying the example we used before, Fig. 3e.

C. Hyperparameter Training

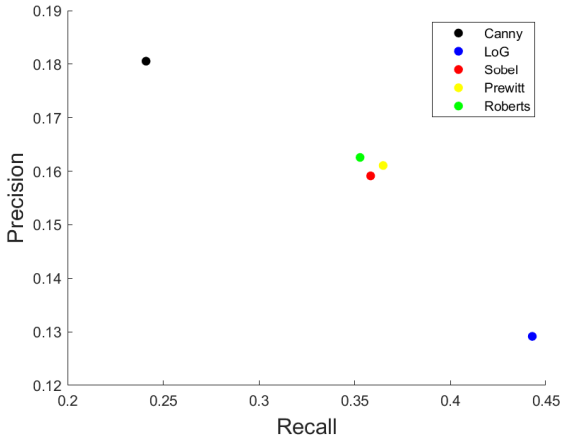
For all detectors except Canny detector, we are supposed to set a threshold to get a binary edge image in final step. However, it is difficult to set a reasonable value for this threshold. So here, besides the regular steps in edge detection algorithms, we apply a threshold training for entire train dataset.

For a specified threshold, we calculate its precision and recall. Then for all thresholds, we get several precision and recall, each pair locates at a point. Finally, we have a precision-recall curve in Fig. 4a. The

[1]https://github.com/DuanRuixiao



(a) Training dataset P-R curve.



(b) Test dataset P-R point.

Fig. 4: PR-curve of train and test dataset. Fig. 4a shows precision-recall curve of training dataset. The recall goes from 0 to 1, but the precision is pretty low, the maximum point is even lower than 0.25. However, the P-R curve is supposed both from 0 to 1 for precision and recall. In this project, we use subject boundary as true boundary, but during processing, we are doing edge detection rather than boundary detection. There must be some differences. Later we will display some images from test dataset and run with our detectors to see the output. And for test dataset, precision-recall point is given as Fig. 4b. Both precision and recall is pretty low. It is hard to say these detectors have good performance.

P-R curve is supposed to be spread both from 0 to 1. However, the curve we plot is lots of different. The recall can be varied between 0 and 1 as threshold changes, but for precision, it is not like what we expected. The maximum precision for the training dataset is still lower than 0.25. But we can explain this issue. Edge is different from boundary. Some edges are not boundaries, but some boundaries are not edges. Since the true boundary we use are from different subjects and they are trying to plot "boundaries" rather than "edges", it is obvious that the result is different and we can not get a high precision when we use this true "boundary" label.

We want both precision and recall be large, and it is the best when they are both 1 under one threshold. Here, we use F-measure mentioned before to find the best threshold. For each threshold, we calculate a F. Finally, the threshold with the largest F is the one we are looking for.

D. Test Dataset Running

In order to evaluate the performances of detectors, we apply these detectors to entire test dataset and calculate the precision and recall which is shown in Fig. 4b. The precision and recall are both lower than 0.5, even far away from 0.5. If we set the threshold randomly, the precision-recall point should have sum above 1. Thus, these detectors have really bad performances. However, consider the true edge image is only "boundary" rather than "edge", it is a little acceptable to get this result.

From precision-recall point figure in Fig. 4a, we conclude that LoG detector works best in these five different detectors, and Canny detector has poor performance. The precision-recall points of Sobel, Prewitt and Roberts locate at almost the same position. These three detectors have similar performances. All items can be viewed both from precision-recall figure and outputs.

Then, for test dataset, we compute the F-measure and show it in Table I. Prewitt has the largest F-measure, which implies it has the best performance. However, all five detectors have F around 0.2. There is only tiny differences between these detectors.

	F-measure
Canny	0.2064
LoG	0.2000
Sobel	0.2204
Prewitt	0.2235
Roberts	0.2226

TABLE I: F-measure of test data set.

E. Test Dataset Examples with Trained Thresholds

Although P-R curve implies that our detectors have bad performance, we still would like to show some example outputs of our detectors. Since Sobel, Prewitt and Roberts have similar F, we only display original image, true boundary image, image after Sobel detector, image after LoG detector and image after Canny detector.

From the examples in Table II, it is easy to find that Canny detector has the least noise. It misses some detail and noise. But the edge is not expected as thin edge. For LoG detector, it shows the widest and brightest edge, which we explained before. Finally, Sobel, Prewitt and Roberts detectors have similar performance, so here we only pick Sobel detector as example. It looks like Sobel detector gives the best performance, edge is thin and clean.

F. Test Dataset Examples with Varied Thresholds

Considered the performance in P-R curve is not good, we are thinking about to change threshold manually and check output images. Since the Sobel detector has the best performance, we here only adjust threshold in Sobel edge detection function. Then display the outputs under different thresholds in Table. III.

When threshold is set at a low value, most of edges will be saved, even some tiny intensity change, like some interior line. As threshold increasing, more edges will be clustered into background. Thus boundary will explore gradually.Finally, when threshold is set at a high value, e.g., 380. Nearly all edges are treated as background. The edges in output images are not continuous and clean.

VIII. Future Research

Actually, the planned topic about this project is "Edge and Boundary Detection", but due to the short length of this summer term, we reduce part of research, concentrating on edge detection. In the following research, we are supposed to study boundary detection based on already detected edges.

Since the true labels we have is only "true boundary" rather than "true edge". We have some problems when we want to benchmark our edge detection algorithms. I believe there will be less problems if we improve the detectors based on current work to do boundary detection.

IX. Conclusion

In this project, we study gradient detection algorithms (Sobel, Prewitt and Roberts), Laplacian detection algorithm (Laplacian of Gaussian) and Canny detection algorithm, and we implement these detectors in MATLAB. Finally, run these detectors with an online dataset and evaluate the detectors' performances.

References

[1] E. Nadernejad, S. Sharifzadeh, and H. Hassanpour, "Edge detection techniques: Evaluations and comparisons," *Applied Mathematical Sciences*, vol. 2, no. 31, pp. 1507–1520, 2008.

[2] G. Shrivakshan and C. Chandrasekar, "A comparison of various edge detection techniques used in image processing," *International Journal of Computer Science Issues (IJCSI)*, vol. 9, no. 5, p. 269, 2012.

[3] O. R. Vincent, O. Folorunso *et al.*, "A descriptive algorithm for sobel image edge detection," in *Proceedings of Informing Science & IT Education Conference (InSITE)*, vol. 40. Informing Science Institute California, 2009, pp. 97–107.

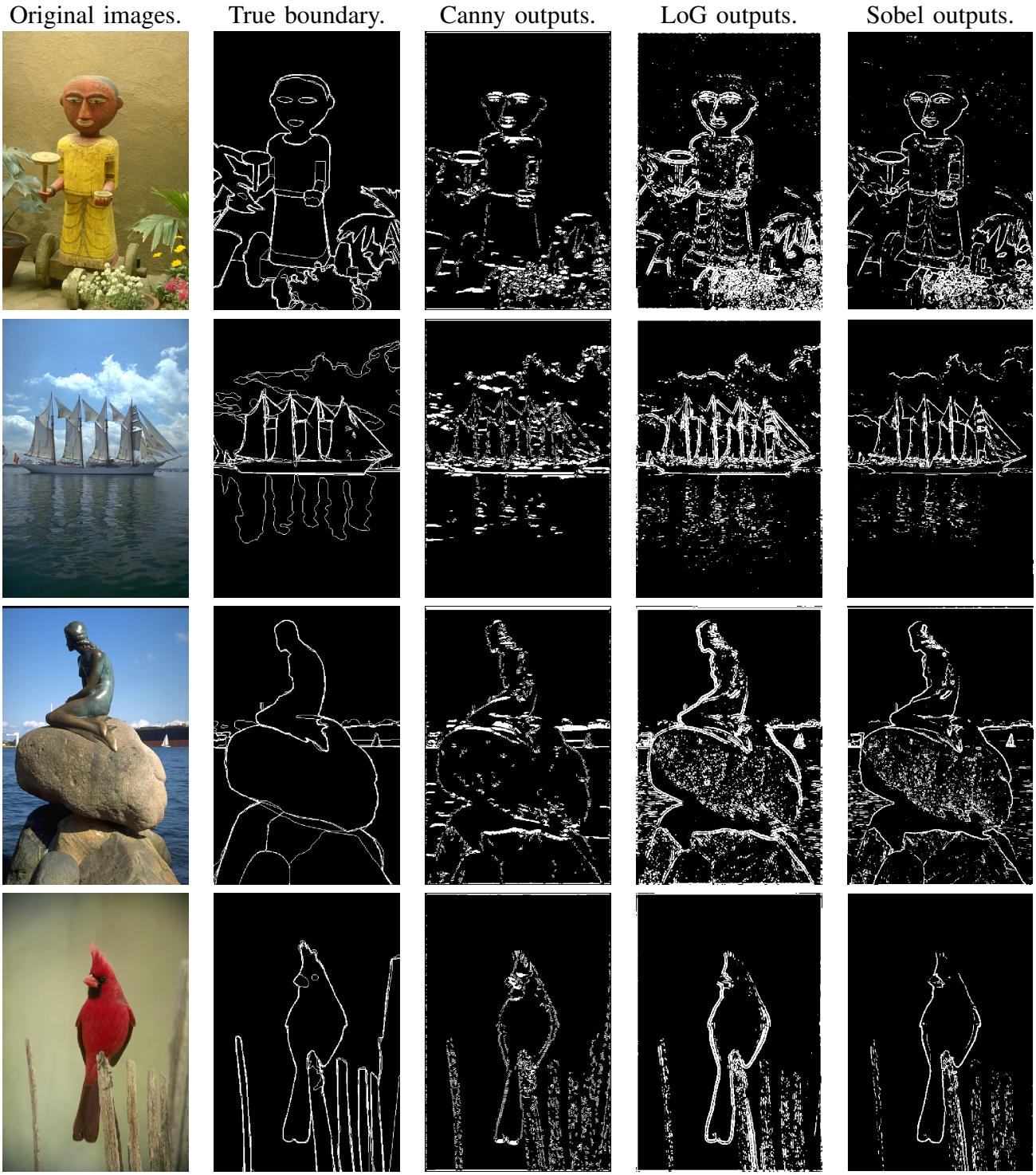


TABLE II: This table shows some examples from test dataset and displays true boundary for each image. Then, following three images which are outputs of Canny, LoG and Sobel edge detector. Generally, Canny detector has least noise, which means it ignores some details. While, LoG detector has the widest and brightest edges. Finally, Sobel detector gives the most clean and thin edge.

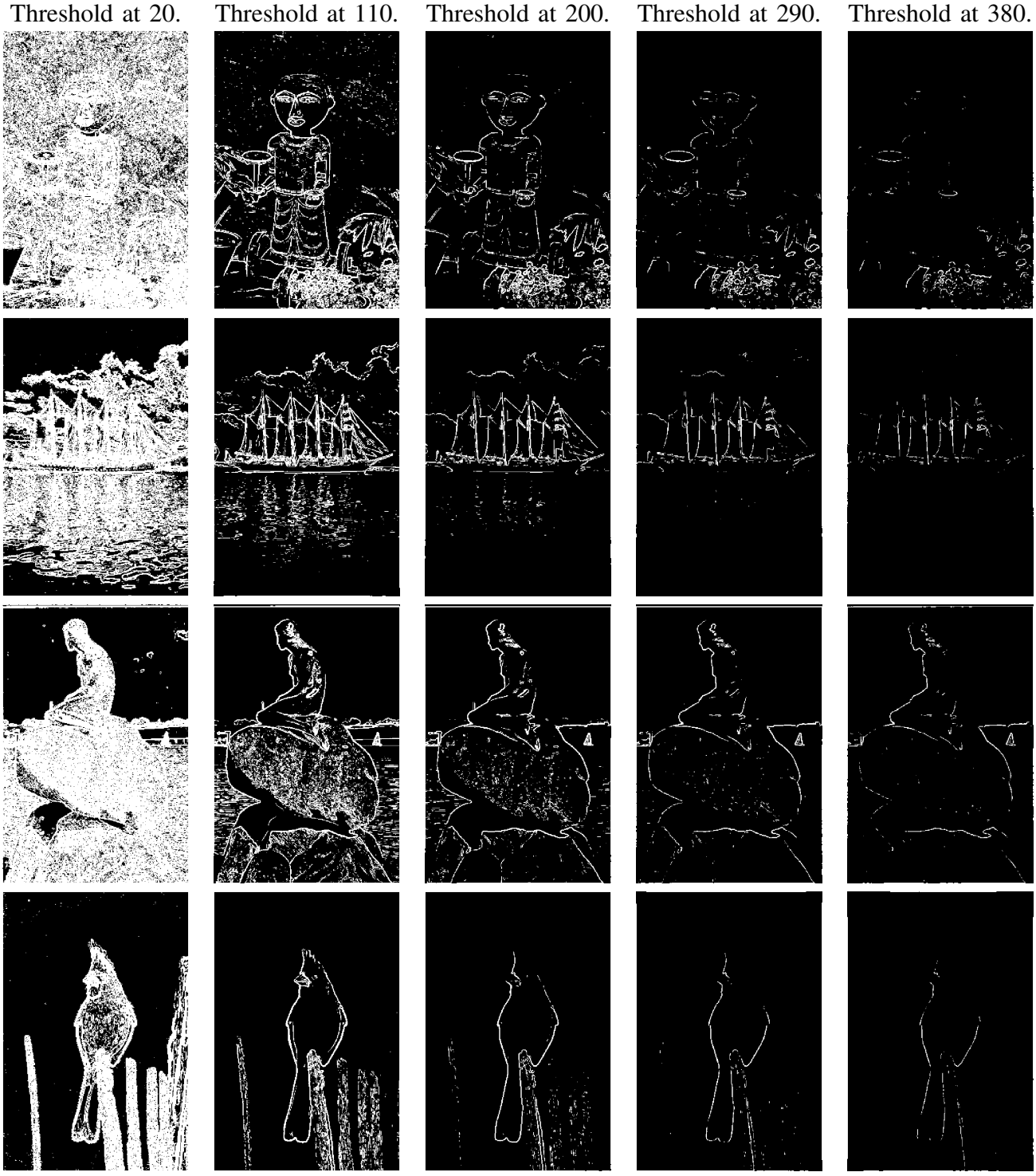


TABLE III: This table shows some examples after Sobel detector under different thresholds. When applying a low threshold, e.g., 20, the output includes lots of details. It keeps tiny edges, then in output it is viewed as noise. As increasing threshold, more light edge will be classified into background. Finally, when the threshold is set at 380, nearly all edges are clustered as background. The edges are not continuous and it is difficult to find objects clearly.

[4] S. Gupta and S. G. Mazumdar, "Sobel edge detection algorithm," *International journal of computer science and management Research*, vol. 2, no. 2, pp. 1578–1583, 2013.
[5] W. Gao, X. Zhang, L. Yang, and H. Liu, "An improved sobel edge detection," in *2010 3rd International Conference on Computer Science and Information Technology*, vol. 5. IEEE, 2010, pp. 67–71.
[6] L. Yang, X. Wu, D. Zhao, H. Li, and J. Zhai, "An improved prewitt algorithm for edge detection based on noised image," in *2011 4th International Congress on Image and Signal Processing*, vol. 3. IEEE, 2011, pp. 1197–1200.
[7] J. Canny, "A computational approach to edge detection," in *Readings in computer vision*. Elsevier, 1987, pp. 184–203.
[8] W. McIlhagga, "The canny edge detector revisited," *International Journal of Computer Vision*, vol. 91, no. 3, pp. 251–261, 2011.
[9] W. Rong, Z. Li, W. Zhang, and L. Sun, "An improved canny edge detection algorithm," in *2014 IEEE International Conference on Mechatronics and Automation*. IEEE, 2014, pp. 577–582.
[10] T. B. Nguyen and D. Ziou, "Contextual and non-contextual performance evaluation of edge detectors," *Pattern Recognition Letters*, vol. 21, no. 9, pp. 805–816, 2000.
[11] L. Er-Sen, Z. Shu-Long, Z. Bao-shan, Z. Yong, X. Chao-gui, and S. Li-hua, "An adaptive edge-detection method based on the canny operator," in *2009 International Conference on Environmental Science and Information Application Technology*, vol. 1. IEEE, 2009, pp. 465–469.
[12] S. Agaian, A. Almuntashri, and A. Papagiannakis, "An improved canny edge detection application for asphalt concrete," in *2009 IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 2009, pp. 3683–3687.
[13] X. Hou, Y. Dong, H. Zhang, and J. Gu, "Application of a self-adaptive canny algorithm for detecting road surface distress image," in *2009 Second International Conference on Intelligent Networks and Intelligent Systems*. IEEE, 2009, pp. 354–357.
[14] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 5, pp. 898–916, May 2011. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2010.161