Massachusetts Institute of Technology

Digital Image Processing (6.344)
Final Project

Edge Detection

Michael Mekonnen

April 30, 2013

EDGE DETECTION

1. Introduction

For my 6.344 final project, I took up the task of edge detection: given an image, produce a new image that clearly outlines the edges in the original image. This is a well studied idea, and there are various methods available. Here, I will explore two of the most widely used methods: the gradient-based method and the laplacian-based method.

2. Gradient-based method

The gradient-based method detects edges in a $2D$ image by finding the analogue of a gradient for the image and then finding the peaks of the magnitude of the gradient. It helps to think about this in $1D$. To find the "edges" of a $1D$ continuous signal $f(x)$, we would first take the derivative $\frac{df}{dx}$ and look for it's optima since an edge in $1D$ is a fast change in $f(x)$ for a small change in x, which is we can very clearly detect by taking the derivative. For a $2D$ continuous signal $f(x, y)$, we can do a very similar thing. Here, we would find the magnitude of the gradient given by $|\nabla f| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$ and search for its optima. When we are working with a digital image, we are not able to take derivatives, so we approximate the derivatives. There are various ways to approximate the derivatives $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$, but one intuitive and standard method is to use filters like the ones shown in Figure 1.

In Figure 2, I present my overall gradient-based edge detection system. Given an image $f(n_1, n_2)$, we first finds its derivatives in the n_1 and n_2 directions exactly by using the filters shown in Figure 1. Next, we compute an approximation of the magnitude of the gradient by taking the square root of the sum of the squares of the two different derivatives. The standard gradient-based edge detection method then proceeds to finding the pixels whose gradient magnitude exceeds some threshold, and these pixels are classified as edge pixels. This steps is usually followed by an edge-thinning step. The choice of this threshold depends on the particular image at hand. In my project, I chose to do something different from using an experimentally found threshold. I first find the maximum gradient magnitude, M. Then, for a given pixel (n_1, n_2), I set its brightness level in the output image to be:

$$255\left(1 - \left(\frac{|\nabla f(n_1, n_2)|}{M}\right)^{\alpha}\right).$$

1	-1	0	1
0	-1	0	1
-1	-1	0	1
	-1	0	1

1	1	1	1
0	0	0	0
-1	-1	-1	-1
	-1	0	1

FIGURE 1. LTI filters to approximate derivatives of a $2D$ discrete signal, in the n_1 and n_2 directions respectively.

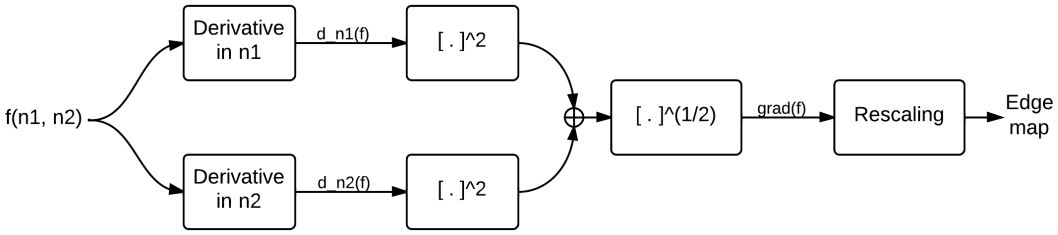

FIGURE 2. Outline of edge detection using the gradient-based method.

What this does is assign pixels whose gradient magnitude is close to M a low brightness (close to 0 or black) and assign pixels whose gradient magnitude is close to 0 a high brightness (close to 255 or white). $\alpha > 0$ is simply a tuning parameter. The resulting edge map traces the edges in the image, and highlights the more drastic edges more strongly. One important benefit of this method is that it avoids having to manually pick a threshold value for each image of interest. There is still the task of choosing the right value of α, but I have observed from experimentation that the same value of α has a uniform effect over different images, so one may choose some default value of α, for example 1, and still obtain a reasonable result for various images. Another advantage of using this method of assigning brightness in the edge-map image is that it avoids discontinuities in places where there are edges that are not highly pronounced. Finally, this method avoids the need for edge-thinning.

3. Laplacian-based method

The laplacian-based method detects edges in a $2D$ image by finding the analogue of the laplacian for the image and then finding the zero-crossings of the laplacian. Once again, let us think in $1D$ to get an intuition for why this is a sensible thing to do. To find the edges of a $1D$ continuous signal $f(x)$, we would first find the second derivative $\frac{d^2f}{dx^2}$ and then look at its zeros, which correspond to the inflection points of $f(x)$. Here we are defining the edges of $f(x)$ to be its inflection points, which is a reasonable definition. For a 2D continuous signal $f(x, y)$, we can do a very similar thing. We would find the

1	1	1	1
0	1	-8	1
-1	1	1	1
	-1	0	1

FIGURE 3. LTI filter to approximate laplacian of a $2D$ discrete signal.

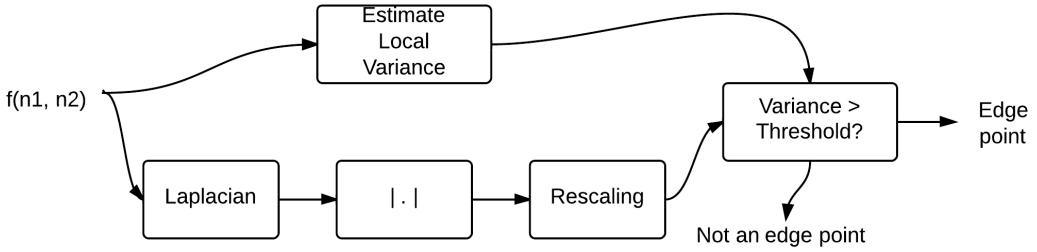

FIGURE 4. Outline of edge detection using the laplacian-based method.

laplacian defined as $\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$ and search for the zero-crossings. When we are working with a digital image, we are not able to take the laplacian, so we approximate it. Figure 3 presents one filter that can be used to approximate the laplacian.

In Figure 4, I present my overall laplacian-based edge detection method. Given an image $f(n_1, n_2)$, we first find the laplacian by using the filter presented in Figure 3. The standard lapalcian-base method would then proceed to finding the "zero-crossings." However, I found great difficulty in doing this and defining what a zero-crossing really means for a discrete signal. However I found an alternative approach. After computing the laplacian, I take the absolute value to get the magnitude of the laplacian. I then rescale these values in a completely analogous manner as the way described in the gradient-based method, but here M is defined to be the 95^{th} percentile laplacian magnitude instead of the maximum. I reached this design decision empirically based on experimentation. It is reasonable to consider high values of the laplacian magnitude to indicate edge points because the points around a zero-crossing point (an edge point) tend to have high values. At this point, we do have an edge map, but this edge map looks very noisy. To get rid of most of the false positives, we compute and use an estimate of local variance at each pixel. If the variance is too low at a pixel, then it is not an edge point. Here, we do need a threshold value to determine a variance that is "too low." I set this threshold to be the 70^{th} percentile variance. Once again, I reached this value by experimentation.

FIGURE 5. Edge detection example: Lena – original; gradient-based; laplacian-based.

FIGURE 6. Edge detection example: Stata Center – original; gradient-based; laplacian-based.

4. Examples

In Figures 5 and 6, I give two examples of the two methods, first on the famous Lena image, and second on an image of the MIT Stata Building.

5. Implementation

I implemented my project completely in the Python programming language. All of my documented code, simple instructions on how to use it, and additional examples can be found at the project repository: `https://github.com/mikemeko/6.344-Project`.